;login:

THE MAGAZINE OF USENIX & SAGE

June 2002 volume 27 • number 3

inside:

CONFERENCE REPORTS

BSDCon 2002

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

conference reports

BSDCon 2002

San Francisco, California

February 11–14, 2002

Summarized by George V. Neville-Neil

This issue's reports focus on BSDCon, held in San Francisco, California, February 11-14, 2002,

Our thanks to the Summarizer:

George V. Neville-Neil

KEYNOTE ADDRESS

Software Strategy from the 1980 Time Capsule

John R. Mashey, Sensei Partners

The opening keynote talk by John Mashey treated us to a time capsule with slides used in the original talks from 20 and 25 years ago.

John's first talk, originally titled "Small Is Beautiful and Other Thoughts on Programming Strategies" was a proposal from 1977 to put a UNIX machine on everyone's desk. The argument for this at the time was that terminal room space was expensive, and that smaller projects, done by teams in their offices, would be more effective.

The talk was really about projects and how they get executed. His basic thesis was that there are three ways of doing projects:

1) complete planning (do it right);
2) pessimism (plan to do it over);
3) build small and quick (the UNIX philosophy).

Some of the more interesting facts he presented were that a survey of projects in 1971 showed that out of 18 total, five were infeasible, five were feasible, but could not gain acceptance, two projects had good fall-outs but no real monetary return, three were partial successes, and three were actual successes. Success is very rare.

He then presented a list of how software goes wrong (which, hopefully, didn't surprise anyone in the audience):

1) too many features (creeping featurism); 2) upwards compatibility; 3) intuition about what's important is usually wrong.

What made this interesting is how little things have changed in 30 years.

The "build it quick" philosophy, which he still holds to, says that you need to be ready to throw a project away and that the purpose of each piece you build is insight into the problem space. Another radical idea in 1977 was to use existing tools.

One suggested practice to keep a project small he termed the "Lifeboat Theory," which says that you need to get rid of a feature to add a new one.

His conclusions were again not surprising: 1) failure is the norm; 2) build fast; 3) keep it small; 4) build it to be changed.

His next talk, from 20 years ago, was called "Software Army on the March," a discussion of project organization for projects of about 200 people. The concept is that there are different types of contributors to a project and they can be organized as an army.

There are scouts on motorcycles who are far ahead of the army. They can fail and are expendable. Instead of building the final product, the scouts should learn how to solve the hard problems and find the problems that the rest of the army isn't even seeing as it moves forward more methodically.

Then there are engineers with bulldozers and cranes who build the actual thing you want to ship (the road, bridge, etc.).

Due to the nature of the audience most of the talk focused on the scouts and on decision making. Decision order (what gets done when) was what he felt was most important.

Questions:

1) What do you see as our failures and successes in software?
UNIX is a success even though Microsoft makes more money. But big coalitions have not been a success. There

was a 1967–77 project to automate Bell Telephone. It cost over $1 billion, and people cheered when it died. It was replaced by a small project run by two managers.
2) Why have we learned so little in 20 years?
There are a couple of answers. One of the biggest is that people are not encouraged to talk about failure. We have a poor institutional memory.
There is a certain amount of pessimism in the philosophy, and that goes against the grain.
3) The talks speak to the human condition. The primary problem is that humans are doing this. Most project management is a way of smoothing over or making them work together better. Our ability to understand things does not keep up with Moore's law.

That's actually another talk that I do, "Hardware, Software, Wetware."

Refereed Papers

SESSION: HARDWARE AND DOCUMENTATION

Porting NetBSD to the AMD x86-64: A Case Study in OS Portability

Frank van der Linden, Wasabi Systems, Inc.

The first technical talk covered a port of NetBSD. The biggest general point was that when programming we should never make assumptions about the sizes of types. The assumption that an integer is 32 bits, which was correct for more than a decade, breaks software on 64-bit architectures. This led to the second general point: a programmer should always expect someone else to use their code on a different platform.

One simple recommendation for building more easily ported code was never to use "#ifdef <arch>". The comment was that "if you use this, there is something wrong with you or the code," and you should go back and check your work.

Questions:

1) How is the optimized syscall different from the normal one?
A bunch of unnecessary checks were removed.
2) Is the way that you do the compilations really separate from the host environment?
In NetBSD we are moving toward a different build system which says that "if you have an ANSI compiler you can build our system." This can now be done, which means we support full cross-builds.

Problems Updating FreeBSD's PCCard System from ISA to PCI

M. Warner Losh, Timing Solutions, Inc.

The old method of updating PCCard support in FreeBSD was to have an adapter that made PCI look like ISA, which allowed the old ISA code to work. But this caused two major problems: there was no IRQ sharing, so the system consumed many IRQs, and it was hard to configure.

Experiences on an Open Source Translation Effort in Japan

Hiroki Sato and Keitaro Sekine, Tokyo University of Science

Hiroki Sato presented work done in Japan to keep up with the mainly English source of documentation in the FreeBSD project. The FreeBSD Japanese Documentation Project (doc-jp) was started in 1996 by FreeBSD developers. Its main goal was translation of FreeBSD documents to Japanese. There are two ongoing efforts: the Japanese Manual Project and doc-jp. Currently, 70% of the documents in the source tree are available in Japanese.

One of the major problems the project has to face is that evaluation of translated documents is harder than evaluating code because you can't run the documents. There isn't a good parser for the output of the translation.

Another issue is release engineering because of the lag time between software/doc release and translation. The team is playing a constant game of catch up.

To help alleviate these problems they have developed a toolchain to process the documents. Unfortunately, the JadeTeX system cannot handle Japanese characters at the moment, so there are no PDF or PostScript versions of the documents available.

There were no questions for this speaker.

SESSION: KERNEL STUFF

Locking in the Multi-Threaded FreeBSD Kernel

John H. Baldwin, The Weather Channel

This talk was intended to show kernel programmers how to use the new locking system and tools. The major tool that the FreeBSD team is using is called Witness and was contributed to the project by the BSD/OS team.

Witness is really a set of macros for locking the store state to ensure that locks are always taken in the same order. An out-of-order set of locks can lead to a deadlock situation, which would be a major failure for the kernel.

One of the beauties of Witness is that it learns on the fly what locking order the software is using. Other systems for detecting deadlocks depend on the programmer to declare the locking order for the whole system.

Questions:
1) Is the granularity of locking on entire processes or are there locks on structures?
This was an example of locking for processes but other locks are possible.
2) Do you have any results on how much time you're spending in the locking code?
Not as yet because the Witness system is very CPU intensive.

3) You mentioned that you were going to go through one of the tools. Can you mention others?
One tool is to use assertions. Another tool on the horizon is a lock profiler to tell us which locks we depend on heavily and where we depend on those locks in the code.
4) What sort of lock push-out has occurred? Are there any subsystems depending on fine-grained locks?
Ninety-five percent of the kernel is still under the giant lock although there has been some work on locking file descriptors.
5) What advice do you have for people maintaining device drivers with regard to pushing locks into the drivers? Are there any recommendations on what to do about spl()?
Leave spl() in there for now. Device drivers are the last thing you want to lock. Subsystems should be locked first.
6) What is the difference between the SMP efforts in NetBSD and FreeBSD?
I'm not completely familiar with NetBSD but from what code I have seen they're using different APIs.
7) You were saying that VM had already been converted.
The only thing that has been done is the file descriptors. We have no performance numbers now.
8) What are you doing about priority inversion?
The mutex code from BSD/OS does include some code to protect against priority inversion.

Advanced Synchronization in MacOS X: Extending UNIX to SMP and Real-Time

Louis G. Gerbarg, Apple Computer, Inc.

This talk discussed Xnu, which is Darwin's kernel and is based on Mach 3.0 and 4.4BSD-Lite2.

The system provides a novel synchronization primitive called Funnels, which are like mutexes but they allow non-reentrant code to run safely. It is not a locking construct. At the present time there are two funnels, one for the network and one for everything else.

Another difference in working with Xnu/Darwin is that it depends on a completely different driver model called IOKit. IOKit abstracts the drivers from the rest of the kernel and provides a stable interface.

Drivers handle multi-threading by using IOWorkLoops. They are used to serialize event sources such as timers and interrupts.

IOKit can also filter interrupt events. This makes it easier to handle a large number of interrupts because they appear to the system as a single event.

Questions:
1) You said that you have a network funnel and a kernel funnel when you read from a socket. Which do you get?
The system calls have flags saying which funnel to get.
2) Do you intend to adopt a particular locking strategy from a particular kernel?
I do not handle policy. The answer is that it's very difficult.
3) It seems to me that the IOWorkLoop that you mentioned is similar in concept to the networking ISR code on *BSD. Is that the same concept?
Yes, they are similar, but the implementation details are different. IOKit is more generic.
4) As more and more devices get attached via USB, does that change your viewpoint on how device drivers should be written?
The USB stack does, in fact, hook up functions to handle asynchronous events instead of depending on interrupts. There is only one IOWorkLoop for all of USB.
5) Have you analyzed the performance of this implementation? What are the interesting effects of the model?
People have done extensive measurements. Just how fast, I cannot say.
6) How many context switches are involved from a NIC card to user space?
I couldn't tell you. Could be less than five, but it is definitely not less than three.
7) Have you benchmarked performance with Darwin vs. something else on the same hardware?
There are answers on the Net. It is not better or worse in totality than anything else.
8) What does the CPU context switch cost?
I don't know.

An Implementation of the Yarrow PRNG on FreeBSD

Mark R.V. Murray, FreeBSD Services, Ltd.

Mark's talk discussed the problem that random numbers aren't random enough.

One of the claimed advantages of this system is that it is number-theoretically correct. The software that was implemented does not block when giving out a random number so it is vulnerable to DoS attacks. Asking for a lot of random numbers does not slow the system down.

Other advantages are that it's resistant to entropy starvation, prediction attacks, as well as being fast and simple to augment.

The target market for this work is the kernel (process IDs, TCP serial numbers) and user-land (SSL/SSH, Kerberos, simulation).

The non-target market is "real" entropy users for things like one-time pads.

The system was designed in an open way from theory to practice.

Questions:
1) Has this been merged from FreeBSD's -CURRENT branch to -STABLE?
No.

2) What is the source of the word "yarrow"?
On *http://www.counterpane.com* take the labs link. A plant with a straw like stalk that is used in China to tell fortunes.

BSD STATUS REPORTS

This session consisted of status reports from representatives of each BSD project. Each section lists the name of the project (OpenBSD, NetBSD, FreeBSD, etc.) the person who spoke, and a laundry list of their latest achievements.

OpenBSD

Todd Miller

- Version 3.0 released in December of 2001 is the 11th release.
- Concentration is on security. Robust code equals safe code. This means using the safe variants of the libc code.
- Integrated cryptography, IPSec, OpenSSH support for hardware crypto
- Much improved UltraSPARC with support
- Alpha resurrection
- Better support for crypto boards Hifn 7951 Broadcom 5820
- Improved 802.11 support
- Support for I2O adapters
- Gigabit Ethernet
- New "pf" packet filter has IPv6 support – easy conversion of IPF-based filters; packet normalization.
- BSD authentication from BSD/OS used throughout the system, including OpenSSH.
- Integrated ALTQ
- Heimdal (Kerberos V)
- Better behavior in low kmem situations
- Updated RAIDframe
- Updated Adaptec AIC7XXX driver
- Better sizing of kernel buffers

Long-term plans:

- IPSec-aware TCP hardware crypto in OpenSSL rate limiting within crypto framework striping network interface

NetBSD

Jason Thorpe

- New toolchain and build framework for fast cross-building
- Highly optimized buffer cache and NFS implementation
- Vastly improved paging behavior
- Performance enhancements for FFS
- Ports to many new platforms (PlayStation 2), including several systems on Chips
- Improved locale support
- Improved standards compliance
- Too many new drivers to count
- Getting ready to branch the 1.6 release

Waiting in the wings:

- devvp: deprecate dev_t in favor of vnodes
- devfs: the end of dev_t as we know it
- wedges: a new approach to disk partitioning
- Sun-style LWPs: finer-grained scheduling
- Scheduler activations: scalable kernel support for threads
- New pthreads library based on scheduler activations
- POSIX real-time extensions
- Support for MIPS64 and PA-RISC
- Performance improvement data structures and algorithms; data movement primitives; concurrency for multi-processors; page replacement algorithms
- Better quality control
- Automated regression testing
- Automated weekly snapshots
- Better bug tracking
- Instant releases
- Embedded
- No persistent storage
- Improved flash support
- Run-in-place support

FreeBSD

Jordan Hubbard

- There are now almost 6500 ports.
- Biggest changes in the last year were social or structural. These included the purchase of Walnut Creek, which was then bought by Wind River and the later spin-off of the FreeBSD team from Wind River.
- The system is still used by lots of large customers (Hotmail, Yahoo, etc.).
- Still doing three releases per year
- More government funding (security)
- Project management is now more work because of the increased number of contributors.
- FreeBSD Foundation has gotten full version of Java for FreeBSD.
- Doc project is doing really well.
- SMP progress continues; a lot of work to be done there.
- KSE (thread scheduler) has reached a milestone.
- SPARC is multi-user but does not self-host yet.
- PowerPC is single user.
- CardBus and PCCard support has taken a leap forward.
- C99 and POSIX are moving forward.
- devfs are used by default.
- Polling network driver support is in -CURRENT and is backported to -STABLE.
- PAM has been cleaned up.
- POSIX ACL support was added to FS.
- Hardware drivers for NVIDIA and other 3D stuff
- Audio drivers improved
- Improved resilience to DoS attacks

Future:

- Wait for NetBSD to get bug tracking right and take it into FreeBSD
- Figure out hosting third-party projects (e.g., SourceForge)
- Continue to work on security infrastructure

BSD/OS

Don Seeley

- Focus on networking and storage
- Snags: tech shakeout, RIFs, FreeBSD debacle, licensing carryover, BSD/OS 4.3 delays
- Wind River is backing BSD in a big way.

In the Pipeline:

- Itasca Release
- Preemptive kernel with locking
- Real embedded release
- Native BSD tools with visionWARE
- Intel IA32, PowerPC targets
- Margaux release
- Tornado IDE
- Cross-development from a Windows host

Future History:

- Building alliances and partnerships
- Spreading BSD through the industry

Darwin

- MacOS X released on March 24
- Targeted the consumer market
- Deployed in a hardened mode
- Released Darwin 1.3 in April
- MacOS 10.1 released in September – a lot of software updates.
- Darwin release with 10.1 was supposed to be 5.0.
- WebDAV is now open source.
- A lot of software is coming out.
- MacOS X will now be the default with every system Apple ships.
- BSD on the desktop now outnumbers Linux by 2:1.
- All hardware comes with developer tools.
- Next event is the World Wide Developer conference, which will be UNIX oriented.
- New release focuses on gcc3.0, FreeBSD synchronization, better pthreads, Kerberos, QuickTime 6, package management.

Questions:

The questions were put to all the speakers, whose answers are noted by project name.

1) I'd like to hear the plan that will stabilize the API for device driver authors.
Darwin: Has its own and it's open source. No plans to change.
BSD/OS: Is going toward the CardBus-based system.
FreeBSD: Most problems are legal and not technical.
NetBSD: Closest to BSD/OS.

2) (Comment) I created the API mailing list after a recent debacle. No one else seems to want to care. The lists are moderated. There is an announcement list and a discussion list:
bsd-api-announce@wasabi-systems.com
bsd-api-discuss@wasabi-systems.com

3) Where lies the official support for Darwin Intel? Is there a plan to ship anything?
Apple will only ship a CD-ROM, no hardware.
4) What's up with the OpenPackages project?
Project on hold waiting for developer time but can be found at *http://www.openpackages.org*.
5) Darwin/MacOS X plans for IPv6?
No formal plans as there is no real demand yet, but the code is in the repository.
6) Four different platforms are doing locks differently. Would it make sense to reduce the divergence?
BSD/OS is synced up with FreeBSD APIs.
Perhaps there should be a forum for SMP.
7) LFS is cool — why not use it?
Although we've made the cleaner a little more robust it's still not ready for prime time.
8) What are the current plans for transferring the FreeBSD trademark to FreeBSD from Wind River?
No comment.

9) Very little seems to happen with merging the user-land. Speaker proposes that we get someone to drive this.
Most people are waiting for automated tools to help with this.
The core problem is a difference in goals.
Best off collaborating on APIs and not on implementations.
It's not always true that multiple implementations are bad.
10) (Comment) At the recent FAST conference there was a paper presented on the write buffering LFS that significantly improved the garbage collection. Overhead was very small.

KEYNOTE ADDRESS

UNIX: Not Just for Geeks Anymore

Brett R. Halle, Director, Core OS Engineering, Apple Computer, Inc.

The second-day keynote was presented by Brett R. Halle from Apple Computer. He made the usual assertion that UNIX has for decades been designed for engineers but is not usable by mere humans.

Most modern UNIX's primary customers remain servers, scientific users, colleges and universities, specialized workstations. These are fundamentally highly educated technical users in centrally managed environments.

What's missing?

General purpose desktop/mobile computing. This means that the current UI is un-acceptable and that applications are important.

He then presented a MacOS X architectural slide that shows how applications are supported independent of the kernel.

Apple found that UNIX on the desktop uncovered many challenges and presented some interesting opportunities. A central question is, "Who uses the desktop?" Apple found that desktop users include PC users (kids and parents), cre-

ative professionals (in advertising, film etc.), and people working in higher education and scientific/technical fields.

In general desktop users have no access to central administration, aren't computer scientists, and want to use the computer as a tool. These users run every type of software imaginable and are in highly dynamic environments.

One of the major challenges to UNIX in this environment is the file system layout. Names like "/etc /usr /bin" do not mean anything to a typical desktop user. Files are often removed accidentally in some environments because it's common for users to clean their disks and remove things they don't think they're using. For this reason MacOS X provides a filtered view of the file system.

Metaphorically this is a "trees vs. forest" argument. The UNIX file system tree is an extremely powerful concept, but desktop users expect to see a forest and not the trees. Another important feature is that devices should be visible as objects in the system as should remote file systems.

Desktop users often identify files with a descriptive phrase used for identification, including spaces and punctuation. To most users case doesn't matter. Another complication is that paths can change because users reorganize their data frequently.

Another area that UNIX does not address on the desktop is security. On a personal computer, users are the administrators. Unfortunately, root is too granular and highly dangerous. In MacOS X, support for administrative roles is much more flexible, and roles may change during a session. All systems are shipped with root turned off.

What makes this all more difficult is that security issues scare most computer users. A personal computer should have most network services turned off by default. Services should come online in the most secure manner. Updating software for security fixes is difficult and so support for this is built in.

File system permissions are another area of security that has to be dealt with. The UNIX permission model is too restrictive for desktop users, and it falls apart with removable media because of the problems with administrating UIDs and GIDs on someone else's desktop computer. File system permissions only work in a managed environment.

Next Brett turned to the role cron and other background services play in maintaining the system. These are typically used for housekeeping, but they don't work well for computer users with portables or energyStar support, which routinely put systems in sleep mode. These services can also have unpredictable side effects.

The software development cycle is also different for desktops. Commands and libraries are part of the public API, not just what's inside libc. Commercial applications depend on these and do not update on the same schedule as OS releases. Unlike open source projects, source compatibility isn't good enough. Customers cannot be expected to update third-party applications for each OS release.

UNIX could improve by avoiding fixed length data such as usernames, passwords, etc. Brett challenged the audience by asking, "What if the API you were working on had to survive 10–20 years? In a binary compatible way?"

He also pointed out that people make products that extend the kernel to such as file systems, VPN, and device drivers. We've got a lot of work to do here!!!

Apple has put a lot of work into user interface issues. A command is not a good interface for a desktop user and neither is X Windows. Consistency is a very important goal for the UI. There are also accessibility requirements for all users.

Interoperability between applications – data interchange, cut/paste, drag and drop – also has to be addressed.

One of the major goals for OS X was to be able to have multiple localizations from a single binary. This has now been achieved.

Other challenges to UNIX on the desktop are:

- Power management
- Systems can be expected to change speeds
- Subsystems will power down disks, displays
- Systems will sleep when idle
- Mobility
- Wireless networking
- Moving from managed to unmanaged networks

In summary Apple believes we in the UNIX community still have a long way to go but that we can surmount these challenges, in part because we now know what some of them are.

Questions:

1) (Comment) There was one thing I think you might not have gotten right. The problem of backwards compatibility has been solved. You can run old NetBSD or Sys V.3 on NetBSD. I think we've been concerned about that as naïve users. All the old syscalls are still there.

I accept that there are areas where this is true, but I'd like to propose a stronger challenge including evolving kernel plug-ins. We still have the command-line problem as it is an API.

2) I do object very strenuously to the Apple thread of bashing UNIX by referring to us as geeks.

I'm sorry you interpreted it that way.

3) With respect to roles, are these distinctions obvious in Darwin?
It's visible in Darwin; there are authorization APIs. There are no setuid apps.
4) Why did Apple choose UNIX with all of its current problems on the desktop?
I think the choice of UNIX for the base system was the right decision. It's the right technology, but we need to solve these broader issues. I think this is the right way to do an OS.
5) There is a distinction between UNIX programs and Mac applications. It's a different way of writing things. You have to be aware of that.
UNIX has this notion that files should be in plaintext ASCII; the Mac does not have this.
With MacOS X we're trying to balance the goals of original UNIX with building a real Mac.
6) The UNIX security model is pretty well understood. How do you go about ensuring that the new models don't have holes?
Good question. As a commercial entity we have more resources to throw at the problem. The model of moving from a most secure system to a least secure one helps.
7) One thing that concerns me is Netinfo.
This will be addressed in a near-future release.
8) When will I get multiple mouse buttons?
I'll give that feedback to our people.
9) What about central administration?
UNIX already does that well.
10) Is there some thought within Apple to get some ideas in this talk out to open source?
Yes, that's why I'm here.

SESSION: FILE SYSTEMS

Running "fsck" in the Background

Marshall Kirk McKusick, Author and Consultant

This paper was one of two given the "Best Paper Award."

The goal of this work is to be able to quickly and reliably restore file systems to a consistent state after a crash. The work is intimately tied to soft updates. In a soft updates enabled file system, the only possible inconsistencies are that there might be blocks or inodes marked in use that are free.

With this in mind it is safe to run immediately after a crash, though, eventually, the lost space must be reclaimed. The problem is the loss of resources during this period.

To address these issues there is background resource recovery. The steps to achieve this are to snapshot the file system and then run standard fsck on the snapshot. Making this happen requires adding a system call to allow fsck to put lost blocks and inodes back into the file system map.

Information on the creation and management of snapshots and background file system checking can be found in the paper.

The status of the work is that snapshots and background fsck have been running on FreeBSD 5.0 (-CURRENT) systems since April 2001. All relevant code is under BSD license and can be found on *http://www.freebsd.org*.

Questions:
1) Can this work be re-used to revive the union mount system?
That's an independent system, so it's unrelated.
2) What are the potential failure modes during snapshot generation?
There is no failure mode since a snapshot is not a snapshot until it's marked as finished.
3) When you're performing copy-on-write is there a failure mode?
There are problems there because you have to write synchronously to make the snapshot consistent.
4) Modern ATA disks have problems with soft updates due to their lying about operations.
If you have disks that lie to you then you're hosed no matter what. The current answer is tagged queuing, which is only in SCSI right now.
5) Since you can adjust the link count on any file, how does that interact with security?
These operations can only be done at security level 1 (root).
6) Your modifications to the scheduler — how do these affect multi-user performance?
By default everyone runs at nice 0, so if you aren't running niced you won't notice it. If you do nice things it will slow you down. It makes nice a CPU and I/O thing now.
7) Have you given any thought to making the whole process FS independent?
This is too tightly integrated and is not generalizable.

Design and Implementation of a Direct Access File System (DAFS) Kernel Server for FreeBSD

Kostas Magoutis, Harvard University

This paper was one of two given the "Best Paper Award."

DAFS is an NFS derivative that provides user-level file access for data-center applications. The protocol itself is specified by the DAFS Collaborative led by Network Appliance and Intel and is targeted for virtual memory mapped networks.

A VM mapped network has user application memory mapped directly into the network interface controller. This gives user-level access to network data with low overhead. It is most commonly used in Server Area Networks (SAN).

The protocol supports both remote DMA and reliable, in order, message passing. Like NFS the protocol is based on Remote Procedure Call (RPC).

Within the kernel the DAFS server talks to the vnode layer for file access. All file I/O goes through the buffer cache. Connection setup is through its own driver. Network data transfer is direct via the NIC. The system uses the PMAP layer of the VM system to register the protocol with the NIC.

As part of ongoing work, they are trying to provide asynchronous vnode file I/O and to integrate NIC MMU directly into the VM system.

For more information: *http://www.eecs.harvard.edu/vino/fs-perf/dafs*
http://www.dafscollaborative.org

Questions:
1) What's the security model for RDMA.
Security is kind of weak right now. At the device level it is possible to corrupt exported pages.
2) This work shows that UNIX I/O is problematic for user space. Perhaps we need a new model.
Conversation taken offline.

SESSION: NETWORKING

Flexible Packet Filtering: Providing a Rich Toolbox

Kurt J. Lidl, Zero Millimeter LLC; Deborah G. Lidl and Paul R. Borman, Wind River Systems

Paul Borman discussed packet filtering in the BSD/OS system. Although their system is named IPFW it has no relation to FreeBSD's ipfw subsystem. The system was developed in the early 1990s when the only technology available was based on rules and not programmatic (screend, IPFirewall, IPFilter, etc.)

BSD/OS's IPFW is a framework for IP filtering that supports multiple types of filters and is agnostic about how filtering is done. It provides a programmatic interface to the system. The primary method of programming is based on the Berkeley Packet Filter. Filters can have names.

IPFW filters packets at several points in the network stack. Each filter point is a chain of filters and each chain can have zero or more filters. Filters can be of multiple types and are normally pushed onto the list like a stack. They can be prioritized and named as well.

Types of filters and some examples are given in the paper. Paul presented these to the audience. The examples showed how to deal with things such as the NIMDA and Code Red worms.

Questions:
1) Is the language static? Is there a compiler?
You need the source.
2) Is this going to be open source?
Currently undecided by their employer (Wind River Systems).
3) Can the system handle IPv6?
Yes.

Resisting SYN Flood DoS Attacks with a SYN Cache

Jonathan Lemon, FreeBSD Project

Jonathan presented work with TCP SYN caches and SYN cookies for resisting denial of service attacks. A SYN-based DoS attack floods the machine with SYN request packets. Each SYN tries to set up a new connection to the machine, and this ties up machine resources, including CPU and memory. Eventually, the machine under attack becomes unresponsive or crashes.

Prior solutions had performance that was a function of the connections in the socket listen to the backlog which is O(N); this consumed 30% of system time. Socket state allocated with incoming SYN was also roughly 512 bytes, and the data accompanying the attacking SYN was preserved.

The SYN cache implementation borrowed from NetBSD, which in turn borrowed from BSD/OS. This implemented a hash table, with limits on the size of the hash chains. The maximum number of entries was also limited so that data within the initial SYN is not preserved.

The hash function used a boot-time secret, so an attacker could not target a specific hash bucket to create a specific DoS attack on the SYN cache itself. The SYN cache only keeps a small amount of state, about 100 bytes.

SYN cookies had advantages over a SYN cache. They allocate no state on the server. The system creates a TCP initial sequence of numbers that is an encrypted cookie returned in ACK, to determine whether a connection is accepted. The ISN is a secret with a finite lifetime. SYN cookies allow an infinitely deep queue of connections.

The SYN cookie system has some problems. It does not allow for the use of any TCP options and cannot handle retransmission of SYNs or ACKs. Other issues are that it is possible for a connection to be accepted with no initial SYN, and it requires a crypto hash on an incoming ACK.

Questions:
1) Do you have any performance comparison to NetBSD?
No comparison is made to other BSDs. The algorithm is roughly identical, but they don't implement SYN cookies.

A FreeBSD-Based Low-Cost Broadband VPN Router for a Telemedicine Application

Gunther Schadow, Regenstrief Institute for Health Care

This talk focused on applying various pieces of *BSD technology to a user installation. The work was funded by the National Institutes of Health to study a next generation Internet as applied to remote medicine. The test system provided for direct physician-patient teleconferencing between 6-9 physicians and a 240-bed nursing facility, serving high-risk, multiply ill patients.

The telemedicine VPN provided H323 video on top of UDP within a wireless network. The physicians' homes were hooked up through cable modems to the rest of the system. Because of the public/private nature of the connections, IPSec technology was used for private data over the public network.

To provide for a decent quality of service the system used ALTQ. This was to prevent starvation of more important signals by outgoing video. Initially camera control was impossible and outgoing audio was useless. ALTQ literally rescued the project.

As routers they used the Soekris net4501. This is a small, cheap ($250), embedded router that can run *BSD and other operating systems (*http://www.soekris.com*).

The router throughput maxes out at 30 Mbps, but this is sufficient for their needs.

Questions:
1) Were there problems with cable modems degrading performance?
Initially we had a 6Mbps downlink. In some neighborhoods we have a problem with going down to 1.5Mbps.
2) On the PicoBSD mailing list someone showed how to use NFS.
I thought about using NFS also but the problem with that is the path length between the systems.
3) You mentioned NAT at some point. How did you get IPSec to get through NAT?
IPFilter will be on the outside and IPSec with be on the inside. You must let IPSec packets through the filter. You trust your tunnel.
I wanted to make it all a package but there is too much fiddling.
4) You mention that you use DNS to get information from the central system? What they have is a certificate. This is what distinguishes boxes/sites. They query for their internal IP overlay addresses using DNS.
5) Is this HIPA compliant?
Yes.

SESSION: SYSTEM ADMINISTRATION

SystemStarter and the MacOS X Startup Process

Wilfredo Sanchez, MIT; Kevin Van Vechten, UC Berkeley

This talk was a discussion of the MacOS X startup process, which is completely different from any of the other *BSDs. The system grew out of work done in NextStep and Rhapsody. Originally it was a hybrid BSD/System V solution.

Some basic problems with the original system were that it depended on lexographic ordering to indicate who ran first. This is fragile. For MacOS they must go into GUI right away and need to launch the display system early.

The new design defines the startup sequence as a progressive bring-up of services. A single startup item describes a service and contains logic to start or stop that service. The SystemStarter manages startup items.

The startup items themselves are contained in the /sys/library directly.

Each startup item contains an executable (typically a script), a StartupParameters.plist, which contains a description, the list prerequisites, and the list of services.

In the future they hope to provide for the partial startup and shutdown of services. This would allow system administrators to bring up all services that X depends on and service X but nothing more.

The implementation provides for parallel startup. Each script is run as its own process via fork/exec. As soon as a script's prerequisites are met, the script is executed. When a script terminates, the list of waiting scripts is reevaluated to look for scripts newly eligible to run.

The new startup system created new needs for the Inter-Process Communication system. Scripts need to display messages, get configuration information, and report success or failure.

To fill these needs SystemStarter listens for messages on a Mach port, and the scripts themselves use a tool to send messages. Messages can go to the console or query for information.

The system is currently deployed on MacOS X and is used by vendor packages.

Questions:
1) Will this be open sourced?
Yes, it's in Darwin.
2) How portable is this?
Have to port part of CoreServices.
3) Documentation?
Not much documentation but there is stuff in the sysadmin manual.
4) Why is there a duplication on the startup messages?
Because plists are going away.
5) Does this deal with things like walking into range of a wireless network?
Right now this only works for boot. There is another state change system called Configuration or something. That's all a little further down the road.
6) How do you handle deadlocks in the dependency change? What if someone messes it up?
The algorithm is dumb. It goes through each item and says, "Are you ready to run?" In your case they would just never get to run.
7) Why do we run /etc/rc to start SystemStarter?
There is a migration plan.

Log Monitors in BSD UNIX

Brett Glass, Glassware

A log monitor is an intelligent agent that automatically responds to conditions revealed by one or more system-log

messages. A log analyzer does the same thing but offline.

A log monitor can detect abnormal usage patterns, recognize abuse, catch worms, detect vulnerability scans, and detect intruders.

The reason for much of this work is that logging via syslogd is now antique.

One of the major applications for this work was the Apache Web server. Apache does not normally use syslogd for logging. Conditional logging in Apache can be done in the configuration files but this is a dirty hack.

The author used SNOBOL4 to implement a powerful worm blocker. The reasons for choosing SNOBOL4 were that it is a powerful string matcher, can call programs to go upstream to inject a firewall rule and can be implemented in 28 lines of executable code.

Some future enhancements are an option to turn off log compression in FreeBSD, integration of algorithms from MIT AI Lab work on determining "interestingness," and a drop in replacement for syslogd, specifically tuned to allow efficient log monitoring.

Questions:
1) (Suggestion) If you want those flags to appear, use the bsd-api-discuss list.

SUSHI: An Extensible Human Interface for NetBSD

Tim Rightnour, NetBSD Project

SUSHI (Simple-to-Use System-Human Interface) is a menu-based sysadmin tool. Application is based on curses and CDK and is easily extensible. The main thrust of the system is around system administration.

SUSHI provides a hierarchical menu structure that can be used to group similar actions together. The system is based around forms (which are a series of questions and answers). Because the system's menus are stored in plaintext, it can be updated without recompilation.

In terms of implementation of the system, the menus are defined by a hierarchy of directories and command files. Forms and menu indices are stored in text files in each directory. Each form can execute arbitrary programs or specific scripts. Scripts can be written in any language or be an executable.

Questions:
1) How does this relate to SMIT on AIX?
Similar but not the same behind-the-scenes.
2) Doesn't this need to be integrated with the rc/system startup?
It's an adjunct.
3) What about having things changed out from under you?
The system reads the rc.conf file and works with what you did.
4) Is this difficult to port to FreeBSD?
No. The only issue might be curses vs. ncurses.
5) Are you planning to make an install replacement with this library?
No, because SUSHI is more of a question/answer system. An installer would be difficult because the user would already have to know what to do.