Our thanks to Murray Kucherawy for his summaries

conference reports

BSDCon '03

September 8–12, 2003
San Mateo, California

Summaries by Murray Kucherawy

KEYNOTE

Computing Fallacies (or, What Is the World Coming To?)

Michi Henning, ZeroC, Inc.

Henning presented fourteen common misperceptions of the technology industry, and explored the fallacies of each. Those of us who have weathered the storm of the dot-com collapse may be nestled in the comfort of stable jobs, but according to Henning, the reality is that we're far from where we need to be and, in some cases, possibly even going in the wrong direction.

Many of these misperceptions involve the idea that computers in the workplace are easy to use and increase productivity. This overlooks some key considerations: Adding computers to the workplace also establishes some infrastructure that has to be maintained. GUIs were expected to close the gap between using or managing complex software and systems, but without truly good GUI designs – and there are very few of those – the gap is only changed, not truly reduced.

Henning asserted that a great deal of computing-related talent is wasted on doing things just because they're cool. This also applies to the latest and greatest word processors and spreadsheet packages. There has been little true advancement in the last decade, but new versions keep coming out, mainly to please shareholders.

Time-to-market pressures have also reduced the average education of a software developer to far below what any seasoned administrator or developer would demand, with obvious detrimental effects. Remember how good you were after just two years? Major subject areas covered 10 years ago, like grammars and finite-state automata, are mysteries to younger programmers. A lot of senior developer productivity is lost explaining debuggers to the fresh crews coming into the market, which hurts both the "bottom line" and our progress in general.

While it is technically true that computers are getting faster, software bloat and inefficiency is completely obscuring the hardware advances offered by manufacturers and their research. Do your favorite Web pages really load any faster? There's less of an emphasis on efficiency; we no longer really care about benchmarks and actual performance comparisons.

How do we deal with all of this and get back on track? Henning says we should start acting in the interests of the people we really work for, i.e., consumers, and not people obsessed with the "bottom line." It needs to be okay for developers to do long-term work, with long-term funding, rather than fussing about how to achieve the current quarter's projections. There also needs to be a code of ethics to quash the high levels of self-interest currently dominating the industry. Changing a single API can cost enormous amounts of time and money. Progress can only come from a lot more cooperation and respect from everyone involved — the market, the developers, their managers, and our sales forces.

STICKY PROBLEMS

Reasoning about SMP in FreeBSD

Jeffrey Hsu, FreeBSD Project

Hsu discussed the logic behind lock placement in the highly anticipated SMP code for FreeBSD. SMP itself is exciting not because it's new, but because it's becoming affordable, making a comparison of the innards of various implementations particularly interesting.

FreeBSD's SMP locking is based on the work done by the BSD/OS team. Only two of the low-level locking primitives are needed in this implementation — namely, mutexes and spin locks. This approach comes from the observation that most locks in the SMP kernel actually go uncontested, so complex locking methods are generally not needed. In fact, it's been observed that bus contention will become an issue before anything complex really becomes necessary. It's a better use of developer time to concentrate on subsystem lock code.

The approach used in SMP locking chiefly depends on what goes into the subsystems involved. There are really only a few places where locks are truly necessary, and other operations should be skipped when considering a locking scheme. User-level race conditions, for example, should really be dealt with out in user space. Locking single atomic reads, e.g., a read of four bytes, would also be a waste of a lock.

Reference counts are also used throughout the FreeBSD SMP kernel. There is rarely a need for an atomic reference count increment/decrement primitive if the basic mutex primitive is fast enough, especially given that most mutexes are uncontested anyway.

Hsu closed by going over some of the basic synchronization concepts that should revive memories of threaded programming courses from years past. Obviously, such practices are especially important in SMP as well, as it is probably a prime example of why those concepts are key.

DEVD – A DEVICE CONFIGURATION DAEMON

M. Warner Losh, Timing Solutions, Inc.

Losh presented his work on devd, an event-driven device configuration daemon package. The goal here is to overcome UNIX's traditionally monolithic approach to devices. Drivers are typically compiled into the kernel or loaded at boot time, but the device subsystems never change while the system is running. The kernel was never designed to have "hot-plug" hardware. With the advent of PCMCIA, USB, Firewire, hot-plug PCI and other upcoming technologies, you can have devices suddenly appear and want to do something.

We don't want to keep writing new daemons for new technologies as they arrive (pccardd, usbd, apmd, etc.). Taking advantage of dynamic kernel loading concepts would be ideal, since it keeps the kernel size down.

The configuration for devd involves defining event-action mappings that can be triggered by, for example, device attach, device detach, and unknown device vents. It is possible to control a device's label even if the probe order changes. Attach events can invoke configuration actions such as triggering dhclient executions, and it is also possible to guide configuration of devices based on location. Device drivers can be loaded when the device arrives, rather than having them built into the installed kernel. The configuration is similar to the format of modern named.conf files to define the event-actions.

Future work will include handling power events, e.g., suspend and resume, dock and undock. Link up/down events will also be able to trigger actions. Also planned is a control socket so that a user-land application can monitor for certain device events.

ULE: A MODERN SCHEDULER FOR FREEBSD

Jeff Roberson, FreeBSD Project

Jeff Roberson took on the task of writing a new scheduler for SMP environments after observing a lack of CPU affinity in the existing scheduler. "CPU affinity" refers to a thread preferring the same CPU for later time slices to take advantage of large CPU caches. Supporting this leads to enhanced support for hyperthreading/SMT (symmetric multithreading) processors. Roberson also observed that the common priority decay algorithms aren't very fair in SMP environments, and current schedulers have no concept of CPUs of non-uniform capacity.

Roberson presented a comparison of various existing scheduler algorithms, including the existing BSD, SVr4, and Linux implementations, before going into the ULE implementation in detail.

The major components of the ULE implementation include several queues, two CPU load-balancing algorithms, scoring of interactive activity, a CPU usage estimator, and slice size and priority calculators. The load-balancing algorithms work together to keep the CPUs evenly loaded under a variety of load conditions, even if the CPUs are of varying power. Since moving cached data regarding a specific thread from one CPU to another carries a cost, migration of threads between CPUs is taken into consideration by these algorithms. Also, threads scheduled for a non-idle CPU can be "stolen" by an idle CPU, and a periodic task evaluates the current load situation and evens it out.

Graphs comparing the performance of the four schedulers under various loads were presented and are available in the white paper.

ULE's gains come mainly from the decoupling of interactivity, priority, and slice size into individual parameters. Other schedulers leave these tightly coupled, with varying side effects. The result of this is a system that appears to be much more interactive even when confronted with a lot of re-niced load: "Livelock under nice load has been a constant problem for UNIX schedulers which ULE now avoids entirely."

RELEASE ENGINEERING

AN AUTOMATED BINARY SECURITY UPDATE SYSTEM FOR FREEBSD

Colin Percival, Computing Lab, Oxford University

Percival's package is intended to address the ever-present problem of lazy system administrators. Though Microsoft is best known for system managers who don't bother to apply security updates in a timely fashion (or, indeed, at all), the open source community is not immune. He asserts that at least 25% of FreeBSD administrators are also behind the curve. Among other things, impediments to improvement involve cvsup being non-intuitive, and the confusing and time-consuming nature of the make-world approach. Some systems lack the resources to make existing approaches palatable.

The trick here is to determine what binaries are affected by a change to a particular piece of source code. The answer is not always obvious. There are some simple approaches one can take, like comparing a make-world result before and after the change, but many files change every time they are built by virtue of such things as build time-stamps. It's possible to work around this since build stamps, for example, are always in the same place, so if that's all that differs, it can be skipped. There are always a few other complications though, such as fortune files, kernel version numbers, and quirks in gcc. Sometimes a particular file will have both crypto and non-crypto versions distributed.

Once a list of files to be distributed has been established, an update index is generated with lines indicating the file to be replaced, the "old" MD5 hash of the file to be replaced, and the MD5 hash of the file replacing it. Updating an old file could have one of several "old" hashes, so each of these is included in the update index, along with a 2048-bit public RSA key, an MD5 hash of the update index signed with the private part of that key, the new binaries, and binary diffs. The distribution of the public key is secured by including its MD5 hash with the updating software.

An obvious limitation to this approach is that the binaries to be replaced must match one of the "old" MD5 hashes published in the update index. Percival says this is a limitation he can accept because such a case would only occur for an administrator who has compiled his or her own source, and such people likely don't have a real need for an automated binary update system.

Of course, this all relies upon trust of the source of the binary updates. This problem can be addressed by Byzantine methods, whereby updates would only be trusted (and therefore installed) if some minimum number of systems independently built and signed the same set of updates.

Building a High-Performance Computing Cluster Using FreeBSD

Brooks Davis, Michael AuYeung, Gary Green and Craig Lee, Aerospace Corporation

The goal of this project is to be able to build a high-performance cluster of machines using commodity PCs, usually running one of the free operating systems. The cluster (named Fellowship, after the Fellowship of the Ring) has four core machines: frodo, the management server; fellowship, the shell server; gamgee, for backups, databases, and monitoring; and legolas, a scratch server with 2.8 terabytes of storage. The cluster as a whole runs FreeBSD 4.8-STABLE, and provides over 183 gigaflops of floating-point compute power (LINPACK benchmark). It consists of 160 nodes, each dual CPU, using a mix of Pentium III and Xeon chips. The network used is GigE, with terminal servers for serial console access, and serial power controllers.

Almost any OS works to build such an environment. Things to consider when selecting an OS are locally available system administrator experience, the applications you plan to run, the maintenance model you want to support, the availability of diskless machine support, and your relationship to the OS vendor for help with all that extra tinkering you're going to want to do. Selection of CPU should take into consideration price, performance and software support, but don't forget to think about dispersing all the heat you will generate!

Network options are again based on operating system support, price, and the needs of your application. You'll also need to tackle the question of public vs. private IPs, for obvious security and provisioning reasons. The node naming and IP assignment convention selected for the Fellowship reflects location in the racks of the machines. Don't name your machines after the services they provide, because this can come back to haunt you later.

Node configuration management can be a headache for clusters of this size. Consider such things as individual vs. automated OS and software installations and network booting. Automation of tasks in a large cluster is critical to efficient use of your time. Also think about your job-scheduling model: manual or batch? Of note here is the Sun Grid Engine (SGE), which has been ported to FreeBSD. Don't forget your monitoring tools (e.g., Nagios (Net Saint), Big Sister, Ganglia, and, again, SGE).

The team learned that in a commodity cluster environment, hardware attrition can be significant, so plan accordingly; investing in neat cabling practices and equipment are well worth the investment; and automation is extremely important.

Future work may include Beowulf-style process management, a checkpoint and restart service, use of a distributed file system (e.g., GFS), on-demand clustering, and a database-driven DHCP service.

build.sh — Cross-Building NetBSD

Luke Mewburn and Matthew Green, NetBSD Foundation

The NetBSD build infrastructure includes the capability to cross-build an entire release, including bootable media. Luke Mewburn presented a discussion of this capability and the changes involved in making NetBSD capable of supporting this system.

Native builds of NetBSD releases don't scale. The number of machines required would be staggering, and the time needed to compile on the slower ones is oppressive. In a cross-compile, one host builds for another architecture, so with a small number of very fast machines, a complete release can be accomplished with greatly reduced cost and time. There is no need for superuser access to do so, even to build the distribution media. A goal in this design was to avoid nonportable OS things like chrooted environments, shared libraries and loopback or virtual file systems. It is important also to separate the build tools from the installed build tools, and to have minimal impact on the NetBSD source tree.

The three main tools used are build.sh, which does the cross-compiles; makefs, which builds file-system images (currently only ffs, but there is planned support for iso9660, ext2fs, and FAT); and installboot, a cross-platform-friendly boot sector writer.

An important feature for the "unprivileged build" process is the change to log-privileged file-system operations, such as permission changes to a "meta log file" instead of actually applying them, and using that information when building installation media and tar files.

The result of this work is a set of regular automated builds for all platforms from a few sources. The system is very simple to use, but it has some teething problems. Not all software is cross-compile friendly.

Upcoming work involves solving the hassles with X11 cross-builds, and some improvement in cross-compiling packages where autoconf is involved, as they seem to have a pattern of not being very cross-compile friendly.

INVITED TALK

Long-Range 802.11 WANs

Tim Pozar and Matt Peterson, cofounders of BARWN

BARWN (Bay Area Research Wireless Network) is a community wireless network based in San Francisco, with access points in the city and one on Mount San Bruno serving South San Francisco, Colma, and Daly City. It is similar in concept to other metropolitan wireless networks such as SFLan, SFWireless, NYCwireless, and SeattleWireless, especially with its common technical and political problems. All of these are built on the concept of member-owned infrastructure, but differ on the for-pay vs. free issue.

BARWN's objectives are to develop and document long-range (over two miles) very low-cost wireless networking, and to provide a test bed for new protocols. Practical applications, such as public safety and incident response, can also serve as a backbone tying together various communities. Other positive side effects involve counteracting the loss of bi-directional expression on the Internet, since BARWN offers limited AUP restrictions, symmetrical bandwidth, no port filtering, and real static address space. BARWN also hopes to bridge the gap between clients on different major providers, so that traffic within San Francisco doesn't need to transit a major network exchange down in San Jose first.

The deployment prefers triangles of coverage, and it has been found that there's good general coverage from the top of Mount San Bruno in the area between Daly City, South San Francisco and Colma, and San Francisco. The sticky legalities of using higher-powered radio frequency repeaters and transmitters, though, can impede deployment. RF radiation dictates how these antennas are deployed, and limits public access to do so. Local governments may even regulate the aesthetics of such deployments. Further, license-exempt devices have no priority rights over any other user.

Naturally there are political obstacles as well. Governments generally own all of the good potential transmitter locations, but work at glacial speeds. Finding the right person and getting full sign-off is no less than a challenge. There are ever-present permit and zoning issues, and finding someone willing to take a risk on a private experimental project is never easy. However, governments do like demonstrations, so BARWN put together a demonstration for a San Francisco Police Department mobile command center using streaming video, and that managed to grease some wheels to get the project where it is today.

Technical challenges also abound. 802.11 doesn't scale well, especially over large distances. Interference with other nearby equipment is also a consideration. 802.11h will go some way to help clear these hurdles using a technique called "frequency co-ordination," selecting the quietest part of the spectrum to use. Build-out over short distance can be done visually, but longer-range installations need to be done with expensive surveys.

FreeBSD has a lot of good, stable support for wireless, with more under development, but Linux is leading the game by a small margin and their efforts tend to work around a lot of firmware idiosyncrasies in various cards more effectively. There is a general lack of drivers for Broadcom and TI devices, although there are unofficial drivers rumored to be out there.

STORAGE/CRYPTO

GBDE–GEOM-Based Disk Encryption

Poul-Henning Kamp, The FreeBSD Project

Kamp's work involves the principle of "making sure data gets lost." User ID and password protection aren't enough for really important data. A hard drive can be easily removed from one machine and inserted in another, mounted as a secondary disk, and read without difficulty. As an extreme example, the battle plan for Operation Desert Storm was stolen from a car on an unsecured laptop!

GBDE is a GEOM-based solution for protection of hard drives with strong crypto. Developed under a DARPA contract, it is file system and application independent, and architecture and byte-endian invariant. GBDE works at the disk level, so an encrypted partition looks like any other partition. This makes it trickier for implementing good crypto, but in the end this approach makes the service easier to use. The invariance is important for media portability, and extends lifetime of the algorithm for future systems.

If an encryption system is too cumbersome, people just won't use it. GBDE, however, is practical and deployable. It uses multiple parallel passphrases, with master key schemes, backup key methods, and destructive keys, which render the data permanently useless when applied. The passphrases are all changeable. The crypto principles applied are all the standard algorithms: AES, SHA2, and MD5. The primary strength of the system is via the crypto, and the secondary strength comes from frustrating attackers via such things as unpredictable on-disk locations and one-time-use sector keys.

The keys used are symmetric, unlike PGP, for example; a 128-bit symmetric key is about as strong as a 2304-byte asymmetric key. Breaking 128 bits of data will open a single sector. Breaking 256 bits will open the entire thing, but you'd also have to try all sectors to find the randomly placed lock sector, and if you try a lot of variant encodings, you'd have to be able to recognize that you have an actual hit in the first place.

The passphrase is the weak point, as usual. To be useful, it has to be long and subtle, using control characters, digits, etc. Of course, people can't or don't want to remember those. GBDE can take a passphrase from anywhere, such as keyboard, USB-key, or chip cards. Kamp recommends making a passphrase out of two parts: your private keyboard stuff and 1–8K of random bits on a USB key, the "something you know" plus "something you have" principle.

Support for destructive keys enables a data owner to get rid of data fast. Kamp's examples included such things as students taking over an embassy, raids on human rights offices by police or college dorms by the RIAA, or perhaps the wife asking, "What takes up those 40GB on our hard disk?" The user can quickly destroy all the lock sectors by erasing the 2048 + 128-bit master key. Attacking the disk now requires O(2^384) work, which is much bigger than the O(2^256) work needed when the keys are intact (though that's a huge amount of work anyway). You get positive feedback that the lock is destroyed. A recover is still possible if the encrypted lock sector can be restored from a backup.

The hit in performance and disk space is minor. The biggest risk is bad sectors, which will unfortunately lock down chunks of the disk.

GBDE is available in FreeBSD 5.0 and later.

Cryptographic Device Support for FreeBSD

Samuel J. Leffler, Errno Consulting

Leffler discussed his work porting the OpenBSD cryptographic framework to FreeBSD and improvements he made in doing so. The goals here were hardware-accelerated cryptographic transformations for kernel and user applications, compatibility with the OpenBSD API, and a pass at tuning for performance. Leffler earned the Best Paper award for this paper.

Core support was converted from SPL-style synchronization to a fine-grained locking method, and the software crypto algorithms were merged with existing ones used by KAME IPSec, the established best-of-breed. OpenBSD API compatibility was always the top priority. Performance of the initial work was slower than desired. There was a lot of extra context switching and CPU use. Leffler was sure things could be faster.

Peak performance of the package was limited by the context switch rate on many systems. The initial framework required two context switches for each operation. Leffler replaced the kernel thread with a software interrupt thread for a vast improvement in performance, by a factor of 3.6.

Making a distinction between normal and "batchable" operations enabled further optimizations. Operations that were not batchable used a direct dispatch method. Replacing the software interrupt dispatch with direct dispatch to the crypto driver was four times faster, so there was already an improvement factor of about 15.

Since many callback methods can take a long time, it was inadvisable to execute a callback method in the context of the device driver. However, the callback used by the /dev/crypto driver does execute quickly and also avoids a context switch. Use of this optimization, with due care in the area of synchronization, yielded another factor of 33 improvement.

All of these improvements reduce overhead on the system, so everyone wins. Comparing the FreeBSD implementation to the one in OpenBSD 3.3 shows that this work yielded more than a 70% improvement for certain hardware up to operand sizes of 1KB. This is now available in the CURRENT and STABLE FreeBSD branches, and NetBSD added it in August 2003. Future work will support asymmetric operations, support for more and better hardware, and load balancing.

Enhancements to the Fast File System to Support Multi-Terabyte Storage Systems

Marshall Kirk McKusick, author and consultant

McKusick presented recent work on extending the capacity of the Berkeley FFS under FreeBSD. The current implementation uses 32-bit block pointers, which means file systems are limited to only a few terabytes. I-nodes lack space to add new functionality, and some newer file-system technology is difficult to apply without changing the existing on-disk format.

UFS2, the new version of FFS, addresses these issues. There is a single code base for both the older and newer implementations. The new on-disk format increases the size of an i-node from 128 to 256 bytes, but directory format is retained. The existing linear scan remains, but there are hooks for an indexing system. The two implementations share directory manipulation code. The idea of cylinder groups is retained, but all geometry information is eliminated. A superblock is added as a superset of the original superblock. The size reserved for the boot block area can be selected rather than being fixed, and a zero-size boot block may be selected for file systems that don't need one.

Extended attributes are now supported. There's optional auxiliary data stored with each i-node, much like Apple data forks. The current implementation allows an expandable 32K that is used to support such things as ACLs and MACs. Timestamp fields are now 64-bits long, and a fourth timestamp has been added called "birth time," which is the actual creation time of the file. The i-node flags are separated into user-settable and kernel-settable flag sets.

I-nodes are now dynamic. Only two blocks of i-nodes per cylinder group are now allocated during newfs, and blocks dedicated to i-nodes are expanded whenever the current i-nodes are nearly exhausted. Some performance enhancements were made to soft updates as well.

Enhancements for live dumps have been added to support snapshots. A setuid root utility has been added to make a snapshot of a file system to allow non-root users to make snapshots. Large file system snapshots are supported.

The amount of memory fsck needs for large file systems is proportional to the size of the file system being checked. Four bytes are needed per i-node, 50 bytes per directory, and one bit per block. This means 1.2MB of memory is needed per terabyte on a file system like /usr, but only 66K is needed per terabyte on an MP3 file system because all the files are very large.

This work has been present in FreeBSD 5.0 for over a year, and has now been ported to NetBSD. Future work will allow extent-based storage allocation by having each i-node store its block size directly.

INVITED TALK

Social and Technical Implications of Nonproprietary Software

Peter G. Neumann, Computer Science Laboratory, SRI International

Neumann presented his views about the open source community and the work it produces in contrast to the industry's commercial output. The BSD community has made significant advances toward high-assurance, trustworthy systems. They are generally secure, reliable, and interoperable.

Where disciplined development is present, open source has many advantages over closed-source proprietary systems, and has the possibility of being very robust. The "best practices" advocated by commercial proprietors are substandard and often inadequate. They suffer from lax requirements, flawed languages, and sloppy development, causing bounds and buffer size violations and vulnerability to viruses. There are some obvious examples of this.

Mass-market software is full of security holes, is bloated and inflexible, and, as Henning pointed out in his keynote speech, is intentionally incompatible even with its own versions. This makes it annoying and expensive to administer. Even worse are closed-box solutions, which can't verifiably satisfy security, reliability, or autonomic operation requirements. This hinders analysis of the system and its implications, impedes improvement of quality, and limits urgent on-site fixes. However, the closed-box approach benefits developers. Hiding intellectual property makes money and encourages consumer retention and loyalty. Reluctant continuance is incentivized. Regrettably, security by obscurity does get some mileage. One would think that closed-box systems have implied liability, except the shrink-wrap user agreement disclaims everything.

Open source has its own problems. The source is available to intruders, and vulnerabilities can be exploited. However, as we all know, open source is also important in promoting the evolution of high reliability and critical systems. Given a really secure system, open source isn't harmful since it's of little help to attackers. Open box solutions, on the other hand, need more work to make them trustworthy, robust, and dependable.

In general, failures are likely due to poor security. Viruses, worms, and trojans are rampant, and there are denial-of-service attacks for which there is, unfortunately, almost no defense. There is a general belief that using cryptography secures you, even if your use of it is sloppy. There is extensive outsourcing of system administration duties, which in itself can be a compromise to security. The Homeland Security Agency wants to deploy Microsoft software globally within its organization to facilitate interoperability. The list goes on.

There can be major social implications for computing failures. The USS *Yorktown*'s engines shut down for nearly three hours because of a Microsoft machine suffering a division-by-zero error. Design issues caused patriot missile inaccuracies. Bad UI design and faulty assumptions caused the Iran Air airbus shoot down. The 1980 ARPANET collapse and the 1990 AT&T nationwide slowdown are also prime examples. Developers of closed-box solutions often make "could never happen" assumptions, eventually with disastrous results.

Again echoing Henning's keynote speech, Neumann also mentioned common development fiascoes, such as rampant feature creep, bloat, incompatibilities, bad requirements, and bad architectures. The update of the nation's air traffic control system left controllers with basically the same equipment and squandered over four billion dollars. The update of the IRS told a similar story.

UI design is also responsible for some major disasters. The design of helicopters that would eventually be dispatched to the Middle East had no requirement for engine shielding against EM interference, or even sand. Pacemakers and antitheft devices do not mix, a requirement that was never considered. John Denver's final flight involved a fuel-starved engine because the user interface for the fuel system in his aircraft was basically "down for right tank, right for left tank, up for 'off'."

Neumann advocates future emphasis on discipline in development and good engineering of products at all levels. We need improvements in evolution, evaluation, education, and training. There needs to be more effort given to responsible operational support. We need open standards for code, interfaces and interoperability, and distribution. There should be progress toward more reasonable contracts, liabilities, and incentives. Sound business models for open-box software need to be designed. Architectures need to be more robust, with minimal dependence on weak components. We must work toward more trustworthy servers, firewalls, and distribution paths. User authentication has to become far less trivial, with bilateral peer authentication. The computing infrastructure has to become a lot more resistant to denial-of-service attacks. We need better protocols and analysis tools.

The discussion that followed acknowledged that many of us involved in the open source movement know and heed these points, but it's an uphill battle. That very day, CSPAN was broadcasting a congressional subcommittee hearing about a recent virus outbreak that was very expensive in terms of both time and money. Microsoft, NAI, and others were invited to testify. The ultimate advice coming out of this meeting was "don't click on PIF attachments" rather than a concession that this is the fault of bad software design. Recent worms have gotten very deep penetration but fortunately haven't been malicious... yet. We've reached a point where large-scale infections are very easy to create. Unfortunately, deaths from such egregious flaws in software design do not get enough attention, and new laws tend to remove liability rather than enforce it.

Neumann says BSD platforms are particularly promising in developing trustworthy systems. An alternative to the commonplace homogeneous Microsoft installations is desperately needed. Open-box software is not the final answer, but it has enormous potential, especially with continued diligence and discipline.

SYSTEM BUILDING

Running BSD Kernels as User Processes by Partial Emulation and Rewriting of Machine Instructions

Hideki Eiraku and Yasushi Shinjo, University of Tsukuba

Eiraku and Shinjo won the Best Student Paper award for their work on this paper.

Running multiple OSes on a single machine enables the simultaneous execution of applications written for different operating systems. There are two typical approaches to this: virtual OSes and user-level OSes. The latter have porting problems, involving tremendous effort and/or detailed knowledge of the host and target kernel and architectures.

Partial emulation of hardware and the rewriting of machine instructions at compile time is done to detect some nonprivileged instructions that are tightly related to privileged ones. Implemented this way, user-level NetBSD is faster than NetBSD on Bochs, a virtual machine implementation, by a factor of 10. Thus, we can generate a user-level OS based on a native system. The success is somewhat limited, though; the source is needed, and it's still slower than NetBSD on VMware or than user-mode Linux, mainly because of the volume of page faults taking place.

There are four key issues to tackle: detect and emulate privileged and some nonprivileged instructions; redirect system calls and page faults to the user-level OS; emulate essential peripherals; and emulate the MMU. The changes to NetBSD to work in this environment were minor but necessary, as unmodified NetBSD 1.6 does not provide the needed facilities. The PTRACE_SYSCALL facility of Linux was introduced, and six constants, including the base address, were changed. After doing this, detailed kernel knowledge was not needed.

To demonstrate their results, NetBSD 1.5 was booted under NetBSD 1.6, and a build of patch was done. User-level

NetBSD and FreeBSD have been generated based on native systems. More than one virtual machine can be run at once.

Source code for this work will be available on SourceForge.

A Digital Preservation Network Application Based on OpenBSD

David S. H. Rosenthal, Stanford University Libraries

Rosenthal presented a "network appliance," a digital preservation system for keeping academic journals published on the Web and accessible over the long term. The goal here is to establish what appears to be a single-function box that can be connected to the Internet with minimal monitoring or administration, and that is cheap to install, maintain, and upgrade.

Rosenthal's solution is a peer-to-peer system of persistent Web caches. The package crawls journal Web sites, distributes to local users by acting as a proxy (keeping available documents that may now be gone), and preserves material by cooperating with other libraries' caches to detect and repair damage. The requirements are low material and personnel costs. As a result of its success since 1999, the system is now in use at over 60 libraries. The original paper covering this work was published at Freenix 2000.

The system uses generic PC hardware, with inherent replication to make it reliable, and open source is used to reduce software costs. Staff costs are reduced by minimizing the need for administration, with a goal of 10 minutes per month.

The first version was based on the Linux router project. Every station ran from a write-locked boot floppy. The kernel and RAM disk file system root was on floppy, and remaining binaries were copied via FTP to a temporary file system. A reboot always returned the system to a known good state. This has gone through about 150 machine-years of testing. Rosenthal has observed that a floppy used only for boot is very reliable. However, a great deal of effort was expended to condense the software down to a single 1.68MB format floppy, and using that format can be annoying.

The second version's design has a few new requirements. It must run from read-only media, but somehow allow fast updates. Signatures against write-locked media must be checked. All disk file systems must be marked noexec. A major OS distribution must be used, with minimized changes to the build process. The OS footprint must be minimal, and everything should be built from CVS nightly. The system trusts only the BIOS and the contents of the boot CD and the write-locked floppy, which contains the entire configuration, passwords, and package verification keys. Everything else can be verified based on this or on data in the key-servers. This design has passed several security audits, takes just over five minutes to boot a single unit, and in a fire-drill test, 96% of all machines in the cluster were upgraded in 48 hours.

Future work involves enhancing the host to add support for DHCP, NAT, native Java, USB storage, and open source BIOS, and for more diverse environments such as governments and developing countries. The source is available on SourceForge.

Using FreeBSD to Render Realtime Localized Audio and Video

John H. Baldwin, Weather Channel

Baldwin demonstrated a FreeBSD box built for the Weather Channel that provides local information overlays onto live feeds. A "Weather STAR," a FreeBSD-based satellite-addressable data device, receives a full feed of weather data, extracts data useful to its location, and overlays it on top of the live broadcast from the studio for rebroadcast locally. What you see on the Weather Channel is a live broadcast showing a weather presenter with national maps, but local forecast information in text over the bottom of the screen.

These boxes can be reconfigured or rebooted via satellite instructions. They support simultaneous NTSC (analog) and ASI (MPEG digital) output. They provide this service for the normal Weather Channel broadcast and also for a new product called WeatherScan, which has no presenters and only relays a local forecast.

One of the changes needed for FreeBSD includes ACPI back-porting to support "soft off," which allows the "power" button to cause a software interrupt, permitting a cleaner shutdown. Several driver updates and fixes were made, and improvements were made to the installation system.

Some problems that need to be overcome: nice is too mean to processes that need large CPU slices; user-land threads have difficulties blocking the whole process with very large read() calls; and there were issues with thread priorities, scheduling, and signals. A lot of this is already addressed in FreeBSD 5.x.

NETWORKING

Tagging Data in the Network Stack: mbuf_tags

Angelos D. Keromytis, Columbia University

An mbuf is a fixed-size buffering scheme used in the BSD network stack. The problem here is that packets require additional attributes for processing, and the available 16-bit flags and the interface information are not sufficient. There are many such potential attributes; IPSec requires four or five, and more appear periodically in other implementations.

Keromytis' work involves mbuf_tags, a dynamically allocated variable-sized attribute buffer, and is similar to NetBSD's aux mbufs. There is a minimal fixed header referred to by an mbuf packet header, and a general memory allocator. A list is added to the mbuf packet header using a most-recent-first method. Kernel modules are free to use their own method, but an API is presented for creating, deleting, appending/prepending, copying, and finding tags. One or two lines needed to be changed for a few existing routines to handle propagation and freeing of tags.

This implementation currently uses malloc() to allocate the tag and its data. Some work had to be done in various network drivers to correct their handling of mbufs.

mbuf_tags can be used to propagate IPSec-related implementation throughout the stack, loop detection for virtual interfaces (with some optimizations produced for multi-threaded kernels), or an improved packet filter engine.

Future work will use the pool(9) allocator to avoid the need for synchronization when allocating memory, tag triggers for use with encapsulation, and application-defined tags.

Fast IPSec: A High-Performance IPSec Implementation

Samuel J. Leffler, Errno Consulting

IPSec is composed of three protocols: AH (authentication), ESP (encryption and authentication), and IPCOM (compression). ESP is the most frequently used. There are also a variety of crypto and authentication algorithms used within these. There are plenty of IPSec implementations, most notably KAME, the OpenBSD IPSec code, FreeS/WAN for Linux, and Linux's IPSec.

So why another implementation? Samuel Leffler sought to implement IPSec with hardware acceleration for FreeBSD and to develop a wireless mesh network. The requirements were support of hardware acceleration, a space-efficient implementation, and compatibility with FreeBSD. The approach chosen was an amalgam of KAME and OpenBSD's work.

Leffler began porting the OpenBSD crypto framework, and cloned the KAME IPSec code onto FreeBSD-STABLE. The KAME code was changed to a callback (continuation) model, and heavy tuning was done. The basic KAME framework was retained for compatibility, but ideas like packet tags, continuations, and code path merging were integrated from the OpenBSD code. The result was familiar to both developer groups.

The performance is dominated by crypto calculations, so several of the bottlenecks that needed to be optimized showed up only with fast crypto hardware. Improvements were made in the areas of the crypto support, reducing processing overhead; data handling, aligning packet data and aggressive coalescing of mbuf chains; network drivers, tackling the usual hardware issues of latency, bus bandwidth, and interrupt coalescing; and system I/O, such as IRQ multiplexing, bus bandwidth handling, interrupt latency, and system effects such as IRQ entropy.

The results: Leffler's Fast IPSec implementation is 60% faster than the OpenBSD code for the software crypto case, and about the same speed as KAME. The peak hardware accelerated operation is more than twice that of any other open source implementation. End-to-end throughput measurements were about 230MBps for a uniprocessor machine and over 400MBps when acting as an IPSec gateway. The CPU and 32-bit PCI are the limiting factors. Future work will include IPv6 support, an overhaul of the PF_KEY code, and an SADB redesign to improve locking.

The WHBA Project: Experiences "deeply embedding" NetBSD

Jason R. Thorpe and Allen K. Briggs, Wasabi Systems, Inc.

Thorpe and Briggs worked with the idea of embedding NetBSD on a host bus adapter (HBA). Typical HBA applications include RAID, SCSI, iSCSI, Fibre Channel and TCP/IP offload. The project of embedding NetBSD onto such a card would allow the offloading of a variety of processing chores from the host server.

The Wasabi Embedded Programming Environment (WEPE) is a merging of user space into kernel space. The entire application lives in the kernel to promote effective interaction with the messaging and DMA hardware. This also permits more direct access to large chunks of contiguous SDRAM. In addition, WEPE is an API for applications that provides portability for user space and the kernel environment, a configuration management framework, and a set of NetBSD kernel modifications.

The API offers file and socket I/O operations and thread and some networking functions. There is also a kshell, which acts as the WEPE debug console, and several kernel environment debugging tools. The operating environment is interesting in its contrasts: The host has plenty of local disk, but the embedded system has no local disk, only a small RAM disk; host reset is under software control, but the HBA reset is not; and applications on the host are largely independent, but on the HBA they are tightly coupled.

Thorpe and Briggs successfully demonstrated an iSCSI target HBA running NetBSD+WEPE in conjunction with Intel and DataCore at Storage Networking World in April 2003. The performance was not as good as they had hoped, but they understand now that their debugging and analysis tools are too limited. Although more work will be needed to move this work from the realm of "doable" to that of "viable," something very useful was produced with many intelligent potential applications.

INVITED TALK

Post-Digital Possibilities

Michael Hawley, MIT

You're never more creative than when you play. Michael Hawley leads a team of graduate student researchers interested in finding out what the next high-tech revolution will be. Photography was the last medium to be subsumed by the digital wave, but Hawley says the digital revolution has hardly started. The biggest theme out of the Media Lab lately is embedded intelligence.

Technology and toys tell us a lot about our advances. Video games are now doing what supercomputers used to do. Even a Furby contains a lot of technology (by old standards) packed into a small toy! There's a lot of infrastructure still to be invented. Lots of it is based on what we already know, but a lot isn't.

Technology in LEDs has improved in recent years. They are brighter, cheaper, efficient, and even networked, with no gels to switch around. But we also like technologies that break new ground, do things we didn't imagine before. They have added chips to coffee mugs, watches, and various other devices. Imagine a coffee machine that knows what you want and how you like it! This can bring about a radical change in interface designs so that a lot of the simple things become automatic.

In the kitchen, such intelligence has been added to various devices to improve automation. They call this "counter intelligence," which produced offers of sponsorship from the NSA and inquiries from the CIA (no, the Culinary Institute of America). They also invented a digital nose, based on some biochemical technology that already exists, which can sound an alarm when it smells that your cake is ready.

Another project involved having a marathon runner swallow a thermometer in a pill (don't worry, it's FAA-approved) and wear a fanny pack full of sensors like a heart monitor, GPS, and a cellular modem to upload it all. They've also added sensors to jewelry to monitor your health over time.

Mount Everest was a testing ground for some biometrics useful when people go to very cold climates and high altitudes. Weather monitoring stations were also deployed and were useful even at extremely low bandwidths. GPS surveying points were set up to monitor tectonic movement of the mountain over time. In Iceland, skiing kinematics were measured, but these need to be as non-invasive as possible while still being able to withstand harsh conditions.

In Hawaii, monitoring stations were built, disguised as tree branches or rocks, for tracking the pollination patterns of some very rare plants that are not fully understood. These need to be completely self-contained as there is no power in the areas they wish to monitor. They also need small monitors to broadcast their observations to bases.

Cambodia is an example of a very poor country that has managed to build good wireless coverage. Orphans with access to a computer become teachers, even celebrities. By local standards, a donated old Macintosh can turn a school into a supercomputing center. The country has a very high percentage of youth, so growing up with access to this technology is obviously very important for their future.

The team also traveled to Bhutan, a country in Asia about the size of Switzerland. It was the last country in the world to get cable TV. While taking photos there, a GPS attached to the team's digital camera would sample their position periodically and add its details to the JPEG metadata, along with lens and camera setting information. This was later extracted when the photos were uploaded, making indexing and describing each photo much easier.

All of the applications presented were remarkable, spanning enhancements both in established technical areas such as photographic journals and in new areas of innovation such as advances in kitchen technology. It will indeed be interesting to see where Hawley's research will take us next.