;login:

THE MAGAZINE OF USENIX & SAGE

October 2000 • volume 25 • number 6

inside:

CONFERENCE REPORT

USENIX 2000 Annual Technical Conference

USENIX & SAGE

The Advanced Computing Systems Association & The System Administrators Guild

conference reports

USENIX Annual Technical Conference June 18–23, 2000 San Diego, CA

Opening Session

Summary by Josh Simon

Announcements

Christopher Small announced that there were 1,730 attendees as of the start of the technical sessions. The program committee received 91 refereed-paper submissions (up from 63 in 1999) and accepted 27 (up from 23 in 1999). Next year USENIX will be in Boston.

Christopher Small

The best paper awards were:

Best paper: “Scalable Content-aware Request Distribution in Cluster-based Network Servers” by Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel

Best student papers (tie): “Integrating a Command Shell into a Web Browser” by Robert C. Miller and Brad A. Myers, and “Virtual Services: A New Abstraction for Server Consolidation” by John Reumann, Ashish Mehra, Kang G. Shin, and Dilip Kandlur

Kirk McKusick, chair of the Freenix committee, spoke about that track. The committee received 56 refereed paper submissions and accepted 29 of them. Most of the papers were open source–related. Because the Freenix track was run with the same rigor as the general refereed papers track, they also presented awards:

Best paper: “An Operating System in Java for the Lego Mindstorms RCS Microcontroller” by Pekka Nikander

Best student paper: “Protocol Independence Using the Sockets API” by Craig Metz

Andrew Hume, immediate past president of the USENIX Association (and now vice president) announced the two annual USENIX awards:

Software Tools User Group (STUG) Award: Tetu Yionen for the initial design and creation of “ssh”, the Secure Shell.

The Flame: Outstanding Achievement Award given posthumously to W. Richard Stevens for his work as an innovator, teacher, and active member of the USENIX community (accepted on his behalf by his sister, his widow, and his children)

Keynote Address

Bill Joy presented the keynote address on his vision on the future of technology.

Based on his 25 years of experience, Joy forecasted the next 25-30 years in computing. He started by looking back at the history: the eventual acceptance of software as research in computer science, the integration of networks and the operating system, and the growth of portability in computing. More and more we’ll see standards defined in English, perhaps passing code or agents instead. He also sees the continued need for maintaining compatibility with open protocols and specifications, noting that protocols often outlive the hardware systems for which they were designed.

Looking forward, Joy believes that Moore’s Law will continue. He expects to see up to a 10^{12} improvement over 30 years based in part on molecular elec-

This issue’s report is on the USENIX Annual Technical Conference held in San Diego, California, June 18-23, 2000.

Thanks to the summarizers:

Doug Fales
Rik Farrow
Kevin E. Fu
Matt Grapenthien
Bob Gray
Josh Kelley
Jeff Schouten
Josh Simon
Craig Soules
Gustavo Vegas

And check out Peter Salus’ impressions of the conference on page 34.

A collection of photographs taken at the conference can be found at *<http://www.usenix.org/publications/library/proceedings/usenix2000/photos.html>*

<http://www.usenix.org/publications/library/proceedings/usenix2000/photos.html>

tronics and improved algorithms. The question of synchronization between different geographies becomes hard when you can store 64 TB in a device the size of your ballpoint pen. We need to improve resilience and autonomy for the administration of these devices to be possible.

Further, he sees six webs of organization of content: near, the Web of today, used from a desktop; far, entertainment, used from your couch; here, the devices on you, like pagers and cell phones and PDAs; and weird, such as voice-based computing for tasks like driving your car and asking for directions. These four would be the user-visible webs; the remaining would be e-business, for business-to-business computing, and pervasive computing, such as Java and XML.

Finally, reliability and scalability will become even more important. Not only will we need hardware fault tolerance but also software fault tolerance. In addition we need to work toward a distributed consensus model so there's no one system in charge of a decision in case that system is damaged. This leads into the concepts of byzantine fault tolerance and the genetic diversity of modular code. We also need to look into the fault tolerance of the user; for example, have the computer assist the user who has forgotten her password.

Refereed Papers Track

SESSION: INSTRUMENTATION AND VISUALIZATION

Summarized by Josh Simon

Mapping and Visualizing the Internet

Hal Burch, Carnegie Mellon University; Bill Cheswick and Steve Branigan, Bell Labs Research, Lucent Technologies

We need tools to be able to map networks of an arbitrarily large size, for tomography and topography. This work is intended to complement the work of CAIDA. So Cheswick, et al. developed tools UNIX-style, using C and shell scripts to map the Internet as well as the Lucent intranet. The tools scan up to 500 networks at once and are throttled down to 100 packets per second. This generates 100–200MB of text data (which compresses to 5–10MB) per day and covers on the order of 120,000 nodes. See <http://www.cs.bell-labs.com/who/ches/map/> for details and maps.

Measuring and Characterizing System Behavior Using Kernel-Level Event Logging

Karim Yaghmour and Michel R. Dagenais, Ecole Polytechnique de Montréal

Karim Yaghmour spoke on the problem of visualizing system behavior. `ps` and `top` are good, but neither provides truly realtime data. He therefore developed a kernel trace facility with a daemon that logs to a file. He instrumented the Linux kernel to trace the events, and then performs offline analysis of the data. The tools do not add much overhead for server-side operations but do add a lot of overhead to intensive applications such as the Common Desktop Environment (CDE). Data is collected up to 500kb per second but it compresses well. Future work includes quality-of-service kernels (throttling the rate of, for example, file opens), security auditing, and even integrating the event facility further into the kernel. Sources are available at <*http://www.opersys.com/LTT*> and are under the GNU Public License.

Pandora: A Flexible Network Monitoring Platform

Simon Patarin and Mesaac Makpangou, INRIA SOR Group, Rocquencourt

Patarin and Makpangou's goal was to produce a flexible network-monitoring platform with online processing, good performance, and no impact on the environment. The privacy of users was also important in the design. They decided to use components for flexibility and a stack model. They developed a small configuration language and a dispatcher that coordinates the creation and destruction of the components. The tool is 15,000 lines of C++, using `libpcap`. The overhead is about 0.26 microseconds per filter per packet. For example, HTTP requests get over 75Mb/s throughput on traces, which translates into 44–88Mb/s in real-world situations, or 600–2600 requests per second. Future work includes improving the performance and flexibility. More details are available from <*http://www-sor.inria.fr/projects/relais*> and released pursuant to the GPL.

INVITED TALKS

Computer System Security: Is There Really a Threat?

Avi Rubin, AT&T Research

Summarized By Rik Farrow

Avi Rubin began his talk by alluding to the February 2000 distributed denial-of-service (DDoS) attacks, saying that those attacks just about made this talk redundant. But perhaps the theme of his talk was that not enough has been learned about mitigating the threats to security, that lessons have not been learned from the past.

Rubin used the Internet Worm to illustrate this point. In November of 1988, Robert T. Morris released the Worm, a relatively small piece of code that used the Internet to copy itself to over 6,000 systems. At the time, those 6,000 systems compromised a large percentage of hosts on the Internet. Damage estimates fell between $10,000 and $97 million. Recovering from the Worm was difficult, as most Internet-connected sites relied on email for communication, and email couldn't get through while Worms were executing.

The Worm used three mechanisms to obtain access to networked systems: a buffer-overflow in the finger server, a backdoor still in sendmail (debug), and the "r" commands. The Worm would crack passwords, using a list of 432 "common" passwords so it could assume other user identities for use with the remote shell.

Today, the finger server has been patched and the debug backdoor commented out (except for short period when a Sun engineer accidentally reenabled it in 1994) of sendmail. But "r" commands are still in use today, and SSH when used as an "r" command replacement (with trust) is still vulnerable to Worm-style attacks.

But the biggest problem then and now was homogeneity. The Worm targeted Sun servers. (A few VMS systems were also affected.) Microsoft systems present the current homogenous environment.

Rubin went on to talk about several recent virus attacks. Rubin had firsthand experience with HAPPY99, as it infected his mom's system. HAPPY99 is very polite and very widespread. It keeps a list of infected systems, can be commanded to remove itself, and does nothing else. It will even replace the DLL (Dynamic Linked Library) it modifies with a copy of the original.

The Melissa virus caused many millions in damages. Named after a dancer in Florida, Melissa first appeared on alt.sex. Because Melissa is written in Visual Basic, it is easy to modify even if you don't know VB, as you can just change its message or various strings, and it will still work (and have a different anti-virus signature as well).

Rubin listed the reasons why Melissa was so successful (besides its promise of a list of hot porn sites):

- Many people use the same mailer (Outlook)
- MSWord on Windows on almost all systems
- Many people click okay to macro warnings
- No separation between MSWord and OS
- Virus protect only works against known viruses

"If every time you received a phone call, there was a chance that an appliance, such as your toaster, could explode, you would likely not put up with it. For some reason, people are willing to tolerate the potential loss of all their data when they receive an email." said Rubin.

There have been other interesting viruses, such as Russian New Year and Bubbleboy. Russian New Year invokes Excel to execute any program on the system. Bubbleboy was the first email virus that did not require an attachment, but did display a dialog prompting the user to delete `update.hta` instead of doing it itself via code, which it could have done. Bubbleboy represents the killer transport mechanism, as it could install any software wanted. Actually, Microsoft makes this easy, as there are system calls that upload files (one call), then execute them (second call).

In summary, Rubin suggested that there were certain deficiencies in our current security model. Besides the homogeneity of desktops, Rubin said, "This seems to be the theme, security by pop-up windows. Do you want to continue? The default is YES." He also suggested the use of Digest Authentication in HTTP to prevent replay attacks, and said that SSL was the right place to do encryption, as the top of the IP stack, where you could tell it is working.

Rubin stated, "Trinoo [a DDoS tool] requires password (people can be very proprietary about their daemons)." He went on to recommend that people remember to make backups, so if something does go awry, you at least can recover.

There were a number of questions. Chris Harrison, Whitehead Institute, asked: "Why haven't we yet seen someone who has caused some really serious damage on a national scale?" Rubin answered, "I don't have an answer to that. Could have been much much worse. We need a sociologist to answer that."

Some one asked about the patch to Outlook. Rubin said that he would install it, but what surprises him is how fast MS produced a patch (440k).

Some one asked two questions. How much risk do you see from polymorphic viruses? How much risk from cross-platform viruses? Rubin answered, "I think polymorphic viruses are harder to detect, making them more dangerous. As for cross platform, right now all viruses and worms are on MS platforms, but when cell phones start including command interpreters, this problem will spread this as well."

Dan Geer, now CTO of @stake (which bought the l0pht) asked: What is your opinion about closing security holes by advertising them? Rubin answered by saying that there are "days I wake up feeling we should advertise vulnerabilities, and other days where I think this is a bad idea." The NIPC (National Infrastructure Protection Council, [I think]) is planning on creating industry forums for exchanging security information and rumors. Rubin said, "We need a way to disseminate information to all of the good guys and none of the bad guys." (laugh)

There were several more questions, including comments about Linux or UNIX users having the inboxes crammed full of impotent email viruses, and a suggestion to use litigation to force better adherence to security practices.

FREENIX TRACK

SESSION: STORAGE SYSTEMS

Summarized by Kevin E. Fu

Swarm: A Log-Structured Storage System for Linux

Ian Murdock and John H. Hartman, University of Arizona

Ian Murdock described Swarm, a log-structured interface to a cluster of storage devices in Linux. Filesystem interfaces are often tightly coupled to the interface of the underlying storage system (e.g., requiring a block-oriented storage device). Moving away from this tendency, Swarm provides a configurable and extensible infrastructure that may be used to build high-level storage abstractions and functionality. The intent was to build more than a just a research prototype.

Swarm is storage system, not a filesystem. However, Murdock presented two filesystems that build upon Swarm. Sting is a log-structured filesystem similar to Sprite's LFS. Ext2fs/Swarm is an unmodified Ext2 filesystem on top of a special logical Swarm device.

Several goals guided the design of Swarm:

- Scalability. As the network grows, scale to demands.
- High performance. Take advantage of the network-disk bottleneck.
- High Availability. Handle failures gracefully.
- Cost-performance. Run on commodity hardware.
- Flexibility.

Swarm moves the high-level abstractions from the server to the client. The server becomes a simple repository for data while clients implement the bulk of functionality. By clustering devices, Swarm removes centralization. In this way, Swarm avoids bottlenecks and single points of failure.

Clients use a striped, append-only log abstraction to store data on servers. This combines log-structured properties with RAID. Each client maintains its own log to eliminate synchronization. The log allows reconstruction of data from server crashes and client failures.

Logs are divided into fixed-size pieces called fragments, which are striped across storage servers. Parity is computed for redundancy. The log is append-only, conceptually infinite, and consists of an ordered stream of blocks and records.

Visit <*http://www.cs.arizona.edu/swarm/*> for more information.

Q: What is the performance to availability tradeoff? A: Write performance is good because Swarm can batch small writes. Large write performance is also good. Read performance is not as good as it should be, but this is an artifact of the implementation. The log structure gives both high availability and high performance.

Q: Is there a mechanism for synchronization of a shared resource? A: We are working on distributed lock service.

Q: How does the cleaner know what is cleanable? A: The cleaner is not aware of what is in the fragments, but it can know what parts of the fragment are alive. Agents participate in the cleaning process.

Q: You might consider a Hierarchical Storage Management (HSM) design instead of cleaning the tapes. A: Cleaning is a very interesting topic we are exploring. Your suggestion might be useful.

DMFS - A Data Migration File System for NetBSD

William Studenmund, Veridian MRJ Technology Solutions

Bill Studenmund talked about the Data Migration File System (DMFS) designed at the NASA Ames Research Center. Implemented for NetBSD, DMFS stores files on both disk and tape. This allows an administrator to transparently archive data to tape without affecting users.

NAStore 3 is a storage system at NASA's Numerical Aerospace Simulation facility. It consists of three main systems: the volume-management system for tape robotics, the virtual volume-management system to avoid small writes to tape by aggregating data, and the data-migration filesystem. Studenmund focused on DMFS for the remainder of the talk.

DMFS is a layered filesystem that places its migration code between kernel accesses and the underlying filesystem. The system works with the Berkeley Fast File System (FFS), FFS with soft updates, and any other filesystem with a fixed number of inodes. The layering decouples development of DMFS from that of any FFS-related development.

To have control over archiving and restoring, there are a number of new FNCTL operations, such as set-archive-in-progress and set-restore-in-progress. Also, a modified `ls` command denotes whether a file is unarchived, archived and resident, or archived and nonresident.

When a user asks for a file stored on tape, a DMFS daemon blocks the user process until the file is sufficiently restored from tape or the restoration fails. However, a user can still interrupt the process by hitting control-C. There is also a utility to force restoration of files from tape into the resident risk. Writes also block until restoration finishes. The archive subsystem stores whole files on tape. Therefore, it is necessary to read from tape to piece together the unmodified parts of a file. To initiate an archival process, one can either run a user utility or let the daemon schedule an archive.

The performance cost of using DMFS is minimal. There is an overhead of 1–2% to access resident files. However, Studenmund jokingly admitted that for obvious reasons `vi` takes about five minutes to start when the data is retrieved from tape.

Q: Looking at HSM, it seems that some files are active, while others are accessed only once a month. Is there a way to mark file as readable but not actually perform the restoration? Say, classify files as active or read once a month? A: This is not supported by DMFS, but sounds reasonably easy to add to the system.

Q: You added system calls to deal with opening/closing files. Why not use a new filesystem with vnodes instead? A: We added only three system calls. Data passed to open is sent to the name-lookup system before the filesystem gets the request. Using a special filesystem that would take file handles as the "names" of files would not work. open(2), stat(2), and statfs(2) take nul-terminated strings to name files. File handles have an embedded length, and can contain nuls. Thus these system calls will not read in the correct amount of data for such a special filesystem to be able to work.

Q: Can you mark files as preferred online rather than archival? A: Everything is done in response to the migration daemon. You could modify the daemon to control this. It is certainly feasible.

A 3-Tier RAID Storage System with RAID1, RAID5, and Compressed RAID5 for Linux

K. Gopinath, Nitin Muppalaneni, N. Suresh Kumar, and Pankaj Risbood, Indian Institute of Science, Bangalore

K. Gopinath discussed the design and implementation of a host-based driver for a 3-tier RAID storage system. The tiers include a small RAID1, a larger RAID5, and a compressed RAID5. The system migrates the most frequently accessed data to the RAID1 tier. The project was motivated by the need for fast, reliable, efficient, quickly recoverable, and easily manageable storage.

Gopinath and his students previously developed the system for Solaris, but are now developing in the context of Linux. The team ran into difficulty with the Linux device-driver framework. Linux has some support to logically integrate multiple disks as one logical disk, but this breaks its own device-driver framework.

Q: A lot of RAID performance is guided by stripe size. How do you support changing stripe size? A: We do not, but we considered issues of the filesystem informing the lower layer (volume manager) about stripe sizes and in the reverse direction about the actual capacities available because of compression.

SESSION: FILE SYSTEMS

Summarized by Kevin E. Fu

A Comparison of File System Workloads

Drew Roselli and Jacob R. Lorch, University of California at Berkeley; Thomas E. Anderson, University of Washington

Drew Roselli presented an analysis of filesystem traces from a variety of environments under modern workloads. The goal was to understand how modern workloads affect the performance of filesystems.

Traces were collected on four workloads: an HP-UX instructional workload with 20 hosts for eight months (INS), an HP-UX research workload with 14 hosts for one year (RES), an HP-UX Web database and server with one host for six weeks (WEB), and a personal computing workload with eight NT hosts for four to nine months (NT). To normalize the measurements, only 31 days of each trace were used. None of the workloads reached a steady state because more files were created than deleted. Drawing chuckles from the crowd, Drew noted that "disk sales confirm this result."

The authors developed a new metric for measuring lifetimes. This involves tracking all files created during the trace and ignoring files created toward the end of the trace window. This measurement reveals that for many workloads, the block lifetime is less than the 30-second write delay used by many systems.

On to the results. Each workload has its own unique shape. Systems with daemons tend to have a knee because of activities such as overwriting Web-browser cache files. Other operating systems tend to either delete immediately or wait until space fills (e.g., emptying the recycle bin). The deletion sweet spot centers around one hour on all but the NT workload. The NT workload reads and writes more than twice the amount of the RES or INS workload. Common to all workloads are that overwrites cause the most significant fraction of deleted blocks, and overwrites show substantial locality. The results also show that a small cache is very effective for decreasing disk reads.

Because many processes memory-map a small set of files, these files are usually cached. The authors also reconfirmed a bimodal distribution where many files are repeatedly written without being read while many are repeatedly read without being written. This buys predictability in post-cache access patterns.

The UNIX traces are publicly available on <*http://tracehost.cs.berkeley.edu/traces.html*>.

Q: How often should such trace studies be done? Can we subscribe to something that collects and publishes such traces? A: If you're thinking of collecting traces, don't do it. Read others' traces instead. Workloads change over time.

Q: Did you filter out access to shared libraries? Or is this included in your trace? A: There isn't much left if you take out the shared libraries, so we kept them in the study.

Q: Are there traces of large sequential flows? This is important to manufacturers. I'm willing to pay for the traces. A: Usually traces do not exist because people are concerned about privacy. [Margo Seltzer from Harvard stood up and said she can help in this area. Contact her offline.]

Q: Are filesystems generally designed with database workloads in mind? A: The Web server has a database in it. IBM compared database workloads with Sprite. There is not a big difference between their results and Sprite.

Q: You mentioned that your tested filesystem is not in steady state. How much is expanding? A: We reported this in paper submission, but we thought it was too wordy so we removed it. I can't remember the growth off the top of my head, but it was somewhere around 30GB.

FiST: A Language for Stackable File Systems

Erez Zadok and Jason Nieh, Columbia University

Erez Zadok discussed FiST, a language for stackable filesystems. Building a filesystem is hard! The kernel is a hostile environment; portability is a pain; development is time-consuming; and maintaining the code is costly. A stackable filesystem makes code extensible and modular. To create a new filesystem, one does not have to remember all the low-level details.

Early research suggested creating new APIs such as the stackable vnodes. However, this suffers from several problems including: modifying each operating system, rewriting existing filesystems, taking a performance hit, using a non-portable API, and having no high-level functionality. A stackable filesystem alone is not enough. It still leaves much kernel work to do.

The FiST approach consists of a language and a set of templates. The language contains simple, high-level primitives while C templates provide kernel-level abstractions. The fistgen program outputs code given a base template (basefs) and a FiST input file. Erez dubbed this system "a YACC for filesystems."

Without stacking, a user system call is translated into a filesystem call. With stacking, there is a translation between the filesystem and basefs. The translation can modify results and arguments. This allows for feature additions such as file attributes and distributed filesystems.

The authors compared the development of FiST-based filesystems with filesystems written from scratch. Four filesystems were implemented under the various models:

- Snoopfs: warns of possible unauthorized access
- Cryptfs: transparent encryption filesystem
- Aclfs: adds ACLs to files
- Unionfs: joins content of two directories

The measured systems include filesystems written from scratch in C, written using the basefs templates, written using the wrapfs templates, and written using FiST.

The authors found that the size of newly written code is one to two orders of magnitude smaller when written with FiST. Development time is shortened by a factor of seven compared to writing from scratch. Meanwhile, the performance overhead ranges between 0.8% and 2.1% to enable stacking.

For more information, see *<http://www.cs.columbia.edu/~ezk/research/fist/>* or email *<fist@cs.columbia.edu>*.

Q: Could you explain the difference in cryptfs results between basefs and wrapfs? Why the strange spike in the graph? A: Wrapfs performs unnecessary memory copies.

Q: How do you handle locking? A: Filesystem templates take care of nasty details like locking and reference counting. We use whatever the VFS provides. Special locking primitives could be added to the FiST language.

Q: What is your intention now that you have built a filesystem framework? Do you intend to write an original filesystem using FiST? A: We have plans to extend the language for low-level filesystems on various storage media.

Q: How do you handle traditional filesystem issues such as consistency and recovery? A: Stacking is completely independent of what happens above and below the stacking. You don't have to know about details.

Journaling Versus Soft Updates: Asynchronous Metadata Protection in File Systems

Margo I. Seltzer, Harvard University; Gregory R. Ganger, Carnegie Mellon University; M. Kirk McKusick, Author & Consultant; Keith A. Smith, Harvard University; Craig A. N. Soules, Carnegie Mellon University; Christopher A. Stein, Harvard University

Chris Stein compared two technologies for improving the performance of metadata operations and recovery: journaling and soft updates. Journaling uses an auxiliary log to record metadata operations while soft updates uses ordered writes to ensure metadata consistency.

Stein discussed three properties for reliability:

- Integrity. Metadata always recoverable.
- Durability. Persistence of metadata.
- Atomicity. No partial metadata visible after recovery.

The metadata update problem concerns proper ordering of operations such that a filesystem can recover from a crash.

There are three general approaches: synchronous writes, soft updates, and journaling. The Berkeley Fast File System (FFS) uses synchronous writes to maintain consistency. That is, an operation blocks until metadata are safely written to disk. Synchronous writes significantly impair the ability of a filesystem to achieve high performance. On the other hand, soft updates and journaling use asynchronous writes.

Soft updates uses delayed metadata writes to improve performance. However, delayed writes alone could change the order of writes. Therefore, soft updates maintains dependency information to order writes. That is, before a delayed write is written to disk, it must satisfy certain dependency constraints. Soft updates is not durable or atomic; has weak guarantees on when updates flush to disk; and requires no recovery process after a crash.

Journaling writes all metadata updates to a log in such a way to guarantee recoverability. It can perform asynchronous with Write Ahead Logging (WAL) to ensure that the metadata log is written to disk before any associated data. After a crash, the system can scan this log to recover metadata consistency. This provides atomicity. With synchronous journaling, the filesystem will have durability. Asynchronous mode results in higher performance at the cost of durability.

The authors performed a microbenchmark consisting of several stages reminiscent of the LFS benchmark. After creating a directory hierarchy, the benchmark writes, reads, and deletes either 128MB of data or 512 files, whichever generates the most files. There is also a test on zero-length files to stress metadata operations.

Macrobenchmarks included four workloads:

- An SSH build (similar to the Modified Andrew Benchmark)
- A netnews server (large working set, no locality)
- The SDET software development environment (timesharing)
- The PostMark ISP benchmark (small file transactions to simulate email and e-commerce)

These benchmarks were applied to two journaling filesystems; a soft updates filesystem, and FFS. One of these journaling implementations has the characteristic that the log is a separate filesystem. This makes it easy to run either synchronously or asynchronously, switching on and off the filesystem's metadata-durability property.

The results show that both journaling and soft updates dramatically improve the performance of metadata operations. However, synchronous journaling alone is insufficient to solve the metadata update problem. Synchronous journaling can penalize performance up to 90% relative to the upper bound of an asynchronous filesystem with no protection. The delayed writes in soft updates improves performance in the microbenchmarks. However, soft updates did not achieve good performance in the netnews benchmark because of the ordering constraints. On the other hand, the delayed writes let soft updates excel in performance on the PostMark benchmark.

By understanding the solutions in terms of the transactional properties they offer, one can derive the relative costs of the properties. The cost of durability is the performance difference between the best-performing solution that does offer this property and the best-performing one that does not. Likewise with the other properties. What one discovers from comparing performance is that the cost of durability is generally large relative to that of integrity. If one sacrifices durability, performance can be very close to that of systems with no metadata protection, while maintaining integrity.

Q: What part of the performance gain is due to the hefty hardware you're using? Which has a better gain, an expensive disk or an expensive CPU?
A: Synchronous systems would perform better with better SCSI drives and rotation. Asynchronous systems would benefit from more memory and faster CPU.

Q: Did you turn off disk caching? For instance, Cheetah hard drives do caching. A: I can't recall off the top of my head.

Q: In your microbenchmark, what were the file sizes and how long short lived were the files? A: There were many file sizes. There are four phases in the microbenchmark. All files were first created, the filesystem was unmounted then remounted, and then the files were read and deleted.

INVITED TALK

Watching the Waist of the Protocol Hourglass

Steve Deering, Cisco Systems

Summarized by Rik Farrow

Steve Deering elected to use imagery and puns to get across his point: that the waist of IP should remain narrow. The basic image was an hourglass shape, where layers four through seven of IP (application and transport) represent the top half; layer three (IP) is the narrow middle, or waist; and various different transport mechanisms fit into the fat bottom half (ATM, SONET, Ethernet, etc.). Deering pointed out the importance of the IP layer – that it isolates the stuff above it from the stuff below it, and that this is what has allowed the Internet to evolve.

IP allows us to create a "virtualized network" to isolate end-to-end protocols from network details/changes. Having a single IP protocol maximizes interoperability and minimizes the service interface. IP means that we can assume a least common denominator of network functionality to maximize the number of networks.

Then Deering asked rhetorically: why worry about the waist of IP? He gave mostly humorous answers:

- It provides for navel gazing.
- It happens at middle age.
- The IP is the only layer I can get my arms around.
- I am worried about how the architecture is now being lost: the waste.
- This theme offers all these puns.

Deering went on to describe some problems with the waist of IP. For example, IP multicast (which he actually had a lot to do with) and quality of service (QoS) have added to the waist. And a mid-life crises, IPv6, has created a second waist (the hourglass image now has two narrow connections). IPv6 doubles the numbers of interfaces, requires a new Ethertype, and changes in application software.

The reason for IPv6 was that the IPv4 address space was being rapidly depleted. Although IPv4 addresses began to be rationed out in the early '90s, Network Address Translation (NAT, RFC 1631) and application gateways (AG) have helped to preserve the IPv4 address space, but at some cost to the original goals of IP.

IP was also threatened by youths. The ATM community had this vision of getting rid of the IP layer, and having end-to-end ATM. ATM did not supplant IP, and instead we wound up with a complicated hybrid and two address plans. On top of this, we had more fattening temptations: layer-two tunneling protocols (PPP, LP2T, PPP over Ethernet). "This is progress?" quipped Deering.

Deering talked about lost features of the Internet, properties that were true but are not anymore:

- Transparency – devices that modify addresses (NAT).
- Robustness through fate-sharing – design a system so you do not depend on more resources than you absolutely must. TCP connectiveness depends on state maintained at either end. Add in NAT, and you lose this.
- Dynamic routing. There are many places routing is constrained. With NAT, an organization cannot provide multiple outgoing paths. And the failure of your ISP today means that you cannot simply have a separate path to your network (another ISP).
- Portable addresses. (Early connectors could move their addresses with them, but now addresses are not portable.)
- Unique addresses (NAT).
- Stable addresses (NATs and DHCP).
- Connectionless service. (If you are behind the NAT, NATs do not support transaction-oriented protocols like UDP; protocols do not have an explicit teardown, so NAT boxes just apply their own heuristics.)
- Always-on service, most users of the Internet are behind dialup services, so they must connect or disconnect from the Internet. PPPoE makes cable/DSL like dialup.
- Peer-to-peer communications, not symmetric devices (no servers behind the Net). It becomes the purview of the ISP to set up servers.
- Application independence is another key feature, as more and more application knowledge has to be built into the NAT. If NAT had been widespread before we got the Web, we might have not gotten the Web (no servers behind NAT).

In case all this didn't make the network manager's job hard enough, we renamed bridges to switches. Router people renamed routers as switches (multilayer switches and layerless switches, but this just obscures what is going on). Is this entropy or evolution? Deering believes that this looks like the normal entropy of all large engineered systems over time. "Of course it grows warts and hats over time," said Deering.

What Deering really wants to do is turn hourglass into wineglass – IP over copper, fiber, radio with no intervening protocols. If the lower layers are making IP jobs harder, then let's get rid of lower layers. And we are seeing signs of IP evolving this way. We saw IP over SONET, but people are asking what value does SONET add to IP? We need IPv6 to get back to the narrow shape. Deering concluded, "This is my dream. Only time will tell."

Rob Pike of AT&T Research arose quickly to ask the first question. Pike mentioned that there is not much flow through the stem of the wineglass. He also stated that he thinks that Deering was not giving enough credit to things like what the phone system does for you, and that he doesn't think that IPv6 can replace this. Deering responded by saying that the shape was not supposed to represent a bottleneck, but a reduction in baggage. Pike responded by saying it doesn't appear that IPv6 addresses many of these issues, such as trunking between sites supported by frame relay. Deering responded that circuits could be better based on routing tables, and that MPLS is the current answer to this problem. At this point, Pike suggested that they continue in private.

Jeff Mogul of Compaq Western Research Labs tried to get Deering to talk about firewalls. (It was obvious he doesn't like them.) Eventually, Deering said that firewalls should be in the end host, not in a dedicated firewall (not that we can throw firewalls away today).

Evi Nemeth asked when we will we see IPv6 really deployed. Deering answered "next year," and when Nemeth asked what will get it there, Deering said, "IPv6 in billions of cell phones will require it."

SESSION: OLD DOGS, NEW TRICKS

Summarized by Bob Gray

LEXICAL FILE NAMES IN PLAN 9, OR, GETTING DOT-DOT RIGHT

Rob Pike, Bell Laboratories

Rob Pike pointed out that for 20 years, the UNIX community has been living with the embarrassing problem that chdir("..") doesn't work properly in the presence of symbolic links. From his paper an example is:

```
% echo $HOME
/home/rob
% cd $HOME
% pwd

/n/bopp/v7/rob
% cd /home/rob
% cd /home/ken
% cd ../rob
../rob: bad directory
```

This annoyance, which causes confusion and headaches, should never have lasted so long. The anomaly exists in most versions of UNIX and Plan 9. With a few hours of work, Pike was able to solve the problem for Plan 9 by implementing an "accurate" path name for every active file in the system. It is guaranteed to be the rooted, absolute name the program used to acquire it. A pure lexical definition of chdir("..") is required. That is, take the current working directory and strip off the last component, yielding the new working directory. However, symbolic links introduce an ambiguity in the pathnames. Rob showed that the "correct" name can easily be determined by context.

Rob explained that in Plan 9, "bind" is like symbolic links and a channel is a filesystem handle. All of the necessary information to disambiguate chdir("..") is present in the kernel. A few lines of kernel code implemented the proper solution. Rob challenged the audience to fix this silly bug in contemporary UNIX implementations. He suggested that we may want to look at the Plan 9 source code, which is freely available now at <*http://plan9.bell-labs.com*>.

GECKO: TRACKING A VERY LARGE BILLING SYSTEM

Andrew Hume, AT&T Labs-Research; Scott Daniels, Electronic Data Systems Corp.; Angus MacLellan, AT&T Labs-Research

Andrew Hume described Gecko – an adjunct system to track the efficiency, performance, and accuracy of the AT&T phone billing system. The fundamental question being asked was: "Is every telephone call being billed exactly once?" Their legacy system comprises dozens of big-iron, MVS computers, running hundreds of programs written in Cobol – it was not feasible to interrupt the operations of this 24x7 production system to add tracking software. However, Hume and his team were allowed to tap into the data flows between processes. The sum of these taps amounted to over 250GB/day.

Given the hundreds of millions of transactions per day, Hume's job was to monitor and verify that billing processing was accurate, timely, and complete. His solution was to create "tags" for each telephone call record at each tap point. Nightly, Gecko would add these tags into a giant (60G tag) database, and produce reports highlighting discrepancies from expected flows.

Hume and his team processed the raw information and analyzed the flows using a 32-processor Sun E10000 and two smaller Sun E5000s. They used a number of basic tools: C, ksh, Awk, and Gre (a special implementation of grep). He compared implementations on the Sun and an SGI Origin 2000, and commented that Gecko relied on a solid SMP implementation and they were stymied by inadequacies in PC environments. He found that the SGI gave a 2.5 price/performance advantage over the Sun. But for huge increases in scalability, they would recommend a cluster implementation.

EXTENDED DATA FORMATTING USING SFIO

Glenn S. Fowler, David G. Korn, Kiem-Phong Vo, AT&T Labs-Research

Kiem-Phong Vo described the Sfio package, which is a faster, more robust, and more flexible replacement for the Stdio package. Stdio has several shortcomings in data formatting. For example, to print an abstract scalar such as off_t on some systems you would use a printf specification of %d – on other systems you would need a printf specification of %ld. The Stdio routines, gets and scanf are unsafe when the length of an input line or string exceeds the size of the given buffer. Stdio has no extension mechanism for printing user-defined types. For example, if you had a spatial coordinate type, Coord_t, you would have to use ad hoc formatting methods.

Sfio fixes the above-mentioned problems. It also contains a generalized mechanism for printing arbitrary bases in the range of 2–64. For example, %..2d will print the decimal value of 123 as 1111011.

Here is an example of how Sfio allows flexibility in printing a integer whose size may vary from architecture to architecture:

```
sfprintf(sfstdin, "%I*d", sizeof(intval),
      intval);
```

Similarly, here is a scanning example for floating point numbers:

```
sfscanf(sfstdin, "%I*f". sizeof(fltval),
      &fltval);
```

In both examples, the sizes of the scalar objects determine their types.

Sfio has hooks for handling user-defined types such as complex numbers. This general mechanism allows user to specify format strings, argument lists, and functions to parse each data type.

Sfio outperforms Stdio on all tested platforms mostly due to the new data conversion algorithms. The code is available from <*http://www.research.att.com/sw*>.

INVITED TALK

Implementing 3D Workstation Graphics on PC UNIX Hardware

Daryll Strauss, Precision Insight

Summarized by Matt Grapenthien

3D hardware for PCs has improved to the point where it begins to rival that of traditional 3D graphics workstations. However, providing these capabilities on commodity hardware poses a number of difficult and interesting problems.

After explaining some of the basic algorithms implemented by various 3D hardware, Strauss addressed several of the challenges 3D presents, such as security issues related to commodity hardware. Also, the scheduling granularity used by the Linux kernel, though more efficient for most processes, is too large for smooth video.

Next, Strauss presented the shortcomings of indirect rendering techniques, and presented an alternative: the Direct Rendering Infrastructure (DRI). The DRI is designed to be secure, reliable, and high-performance, by utilizing the available hardware as much as possible. In addition, the DRI is highly modular, allowing different implementations to use only the subset of software they wish.

The DRI is included in XFree86 4.0, and is starting to be used by 3DLabs, HP, and IBM, among others, even on high-end hardware. Precision Insight's work to provide completely open-source solutions to the problems above has shown great promise, even in this somewhat early stage.

FREENIX SESSION: FILE SYSTEMS

Summarized by Kevin E. Fu

Porting the SGI XFS File System to Linux

Jim Mostek, Bill Earl, Steven Levine, Steve Lord, Russell Cattelan, Ken McDonell, Ted Kline, Brian Gaffey, and Rajagopal Ananthanarayanan, SGI

Russell Cattelan talked about the work necessary to port SGI's XFS filesystem to Linux. In particular, he discussed the filesystem interface, the buffer caching, and legal issues. XFS is a highly scalable, journaling filesystem available for free under the GPL.

To maintain atomic updates to the filesystem metadata, XFS keeps a log of its uncommitted actions. Should the machine crash, XFS simply replays the log to recover to a consistent filesystem. Recovery time depends on the size of the uncommitted log. This is in stark contrast to the `fsck` tool, where recovery time depends on the size of the entire filesystem.

A single XFS filesystem can hold as much as 18 million terabytes of data and as much as 9 million terabytes of data per file. XFS is an extent-based filesystem. That is, data are organized into arbitrarily long extents of disk rather than fixed-sized blocks of disk. This allows XFS to achieve a throughput of 7MB/second when reading from a single file on an SGI Origin 2000 system.

XFS uses the vnode/VFS interface. However, Linux does not use this interface. As a result, SGI created the linvfs interface, which maps the Linux VFS interface to the VFS layer in IRIX. At a small overhead cost in translation, this keeps the core XFS code portable.

Second, SGI added pagebuf to Linux. This is a cache and I/O layer to provide most of the advantages from the cache in IRIX. Pagebuf allows pinning of metadata buffers, delayed writes via a daemon, and placement of metadata in a page cache.

Russell also explained the legal process of encumbrance relief. SGI determined, for all the XFS code, which code was original and which code came from third parties. Most of the original XFS code came from SGI. However, XFS contains some software from third parties. A homebrew tool compared every line of code in XFS against the source code from third parties. After removing the third-party code, SGI released XFS under the GPL.

For more information, see <*http://oss.sgi.com/projects/xfs/*>.

Q: Is preallocation fast? A: Yes. You're preallocating an extent of space, not individual blocks. You're not zeroing out files, but you will receive zeros if you read from preallocated space.

Q: How does preallocation work on an API level? A: There are two ways to preallocate, with a command-line utility or new IRCTL calls.

Q: Could you describe what you did at the vnode/VFS layer? A: Our vnode is now part of the inode. The Linux inode would not work with our design. We mostly map functions from linvfs functions to vnode functions.

LinLogFS – A Log-Structured File System for Linux

Christian Czezatke, xS+S; M. Anton Ertl, TU Wien

Christian Czezatke described the LinLogFS filesystem. Started in 1998, LinLogFS aims to achieve faster crash recovery than that of the Ext2 filesystem. LinLogFS also enables the creation of consistent backups while the filesystem remains writable.

LinLogFS evolved from the Ext2 codebase to a log-structured filesystem with better data consistency and cloning/snapshots. It guarantees in-order write consistency.

If you modify part of an Ext2 filesystem during a backup, the change may or may not be noticed. Worse, a simultaneous backup may only detect parts of the change. LinLogFS does not suffer from this problem because backups can use a snapshot of a filesystem. In this manner, backups are completely atomic while still allowing read-write access to the filesystem.

In the future, Czezatke plans to finish the cleaning program, use less naive data structures and algorithms, and cooperate with the object-based disk project for mirroring.

See
<http://www.complang.tuwien.ac.at/czezatke/lfs.html>
for the code.

Q: Have you reviewed the LFS code for the *BSD operating systems? Can you comment on differences or code overlap? A: We looked at BSD LFS and Sprite LFS. We had slightly different goals. We wanted an LFS for Linux. And the Sprite and BSD LFS goal was higher speed than FFS. Our goal was better functionality at the same speed as Ext2.

UNIX File System Extensions in the GNOME Environment

Ettore Perazzoli, Helix Code, Inc.

Ettore Perazzoli spoke about filesystem extensions in the GNOME environment. Perazzoli is an open source software developer and contributor to the GNOME project. GNOME extends the functionality of the UNIX filesystem by using a user-level library called the GNOME Virtual File System.

There are many file formats to juggle: zip, tar, gzip, etc. Each format uses a different tool to view the file contents. GNOME avoids this problem by using a global filesystem namespace. This allows GNOME to identify and view contents of container files such as tar files. Names in the GNOME VFS use an extension of the familiar Web Uniform Resource Identifier scheme.

Ettore discussed how Microsoft Windows has the "My Computer" concept where one can find all system resources such as files, a control panel, printers, and the recycling bin. GNOME VFS addresses the "My Computer" problem by providing filesystem abstractions, and the asynchronous file I/O problem by implementing an asynchronous API. Asynchronous behavior is important because a GUI should be responsive all the time. If the GUI were to block on a file operation, the user cannot stop the operation. GNOME implements asynchronous behavior through an asynchronous virtual filesystem library.

For more information, visit *<http://www.gnome.org/>* and *<http://developer.gnome.org/>*.

Q: Why did you choose a pound sign for subschema? A: We inherited this from Midnight Commander. You can always quote the character.

Q: How do you handle symlinks with ".." in the path? A: Symlinks only work within their context. For instance, a symlink inside a tar file will only make sense within the context of the tar file.

Q: Have you considered POSIX AIO as twisted as it may be? A: Yes, but POSIX AIO only lets you make reads/writes from/to the filesystem asynchronous, so it does not help in complex cases such as the .tar or .zip ones, when you have to do other stuff besides physically reading the file.

SESSION: DISTRIBUTION AND SCALABILITY: PROBLEMS AND SOLUTIONS

Summarized by Josh Kelley

Virtual Services: A New Abstraction for Server Consolidation

John Reumann, University of Michigan; Ashish Mehra, IBM T.J. Watson Research Center; Kang G. Shin, University of Michigan; Dilip Kandlur, IBM T.J. Watson Research Center

Virtual services (VSes) provide a way to partition resources on a server cluster between competing applications. VSs can be used to set resource limits and to charge resources out to different virtual services. For example, Web servers for two different companies can be run on a single host by treating them as two separate VSes. Such partitioning has traditionally been done by partitioning a physical host into several virtual hosts. Although this approach provides good insulation between services, it does not allow sharing common subservices between co-hosted sites. VSes address this problem by dynamically adjusting resource bindings. Each request's VS resource context is tracked as the request is handed off across application and machine boundaries, thus allowing VSes to remain insulated from each other while still accessing shared applications and machines.

Implementing VSes involves modifying the operating system's scheduler to partition the CPU. A VS also hooks into various system calls (such as process creation calls and network communication calls) via gates in order to track the propagation of work. Gates adjust resource bindings during system calls (binding a process to a different virtual service as needed) and enforce resource limits. These gates are implemented as loadable kernel modules.

Evaluation shows that virtual services have a minimal negative impact on performance and successfully insulate competing services from each other, even when these services rely on a common shared service. VS support is currently available for Linux 2.0.36; support for Linux 2.2.14 is in development. Source code is available from
<http://www.eecs.umich.edu/~reumann/vs.html>.

Location-Aware Scheduling with Minimal Infrastructure

John Heidemann, USC/ISI; Dhaval Shah, Noika

With the increase in use of laptop computers, some way to specify context-dependent configurations and activities could provide a great deal of convenience for users. For example, a user might wish to specify "print to the printer in room 232 when I'm at work" or "disable this background process while I'm on battery." lcron is a context-aware cron that provides a general solution to scheduling such context-dependent activities.

John Heidemann

Context information could come from many sources: GPS receivers, wireless base station name, idle status, network addresses, or battery status are all possibilities. Lcron currently supports the latter two.

The interface to lcron is similar to that of the standard cron and at utilities. A context-dependent `at` command, for example, could be invoked as `at 7pm @work`. lcron allows users to specify mappings between terms meaningful to the user ('@work') and information readily available to the computer (latitude, longitude, router IP address).

lcron has been in use for two years. It is available from
<http://www.isi.edu/~johnh/SOFTWARE/XCRON>.

Distributed Computing: Moving from CGI to CORBA

James FitzGibbon and Tim Strike, Targetnet.com Inc.

In 1997, TargetNet deployed a banner-ad delivery system using CGI. As the network grew, basic CGI showed itself to be incapable of keeping up with the growth. They wanted a solution that was faster than basic CGI, was freely available, would support distributed computing, and would work with other products. CORBA was chosen as the solution that best met these criteria.

The system that was developed involves two main components. First is an HTTP-to-CORBA proxy. This proxy, called the dispatch server, takes HTTP requests from the Web servers and presents them as CORBA requests to the application servers. This dispatch server allows a three-tier architecture (Web servers, application servers, databases) in which only the Web servers are exposed to the Internet.

The second main component is a method of notifying each dispatch server of the available application servers. A service heartbeat daemon serves this role by maintaining a list of available application servers, providing a client with an available application server when queried, and coordinating with other heartbeat daemons. This design avoids any single point of failure and provides excellent scalability.

CORBA offers an open source, distributed object model, with wide language support. The combination of a dispatch server and a heartbeat daemon allows a three-tier architecture with excellent reliability and near-linear scalability.

INVITED TALK

The Microsoft Antitrust Case: A View from an Expert Witness

Edward Felten, Princeton University

Summarized by Josh Simon

Disclaimer: Neither the author of this write-up nor the speaker is a lawyer.

Dr. Felten was one of the expert witnesses for the U.S. Department of Justice in the recent antitrust case against Microsoft. In his talk he discussed why he believed the government chose him, and he explained the role of an expert witness in antitrust cases.

In October 1997 Felten received an email message from an attorney in the Department of Justice asking to speak with him. After signing a nondisclosure agreement (which is still binding, so he advised us there were some aspects he simply could not answer questions on), and over the course of several months, he spoke with the DOJ until, in January 1998, he signed a contract to be an advisor to the case.

What was the case about? Unlike media portrayals, the case was not about whether Microsoft is good or evil, or whether or not Bill Gates is good or evil, or whether Microsoft's behavior was good or bad. The case was specifically about whether or not Microsoft violated U.S. antitrust laws.

A brief discussion of economics may be helpful here. Competition constrains behavior. You cannot, as a company, hike prices and provide bad products or services when there is competition, for the consumer can go to your competitors and you'll go out of business. Weakly constrained companies, or those companies with little or no competition, have what is called monopoly power. Monopoly power in and of itself is not illegal. What is illegal is using the monopoly power in one market (for example, flour) to weaken competition in another market (for example, sugar).

The U.S. government claimed that (1) Microsoft has monopoly power in the personal-computer operating-system market; and that (2) Microsoft used its monopoly power to (a) force PC manufacturers to shun makers of other (non-Microsoft) applications and operating systems; (b) force AOL and other ISPs to shun Netscape's browsers, Navigator and Communicator; and (c) force customers to install and use Microsoft Internet Explorer. These issues are mostly non-technical and specifically economic. Dr. Felten focused on the technical aspects.

Under U.S. antitrust law, tying one product to another is illegal in some cases. For example, if you have a monopoly on flour, you cannot sell flour and sugar together unless you also sell flour alone. You cannot force customers to buy your sugar in order to get your flour. Similarly, the government argued, Microsoft cannot tie Windows 95 (later, Windows 98) together with Internet Explorer unless it offers both the OS and the browser separately. Microsoft claimed technical efficiencies in bundling the products together.

This boils down to two legal issues. First, what was the motive in combining the OS and the browser? The answer to this is provided by documentation and witnesses, subpoenaed by the government, and not technical. Second, does the combination achieve technical efficiencies beyond that of not combining the two? The answer to this is experimental, technical, based in computer science, and was the focus of the rest of the talk.

Specifically, how did Felten go about testing the efficiencies or lack thereof? He started by hiring two assistants who reverse-engineered both Windows and Internet Explorer. (Note that this work, because it was done on behalf of the government for the specific trial, was not illegal. Doing so yourself in your own basement would be illegal.) After nine months, they were able to assemble a program to remove Internet Explorer from Windows.

The next step in the process was to prepare for court. In general, witnesses have to be very paranoid, nail down the technical details, have sound and valid conclusions, and learn how to be cross-examined. Lawyers, no matter your personal opinion of them, are generally very well-schooled in rhetoric, terminology, and framing of questions, and hiding assumptions in them. They're also good at controlling the topic, pacing the examination, and producing sound bites. In his testimony, Felten demonstrated the "remove IE" program. Jim Alchain, Microsoft vice president, provided 19 benefits of tying the products together and claimed the removal program had bugs. In the government's cross-examination of Alchain, he admitted that all 19 benefits were present if IE was installed on top of Windows 95, and that the video used to show that the demonstration of the removal process had bugs had errors and inconsistencies. Microsoft tried a second demo to show problems with the removal program under controlled circumstances and could not do so. Furthermore, in rebuttal to Microsoft's assertion that the products had to be strongly tied together to gain benefits, the government pointed out that Microsoft Excel and Microsoft Word were not strongly tied and yet were able to interoperate without being inseparable.

Judge Jackson reported in his findings of fact in November 1999 that the combination had no technical efficiencies above installing them separately, Internet Explorer could be removed from the operating system, and tying the browser and the operating system together was, in fact, illegal.

The next phase of the trial was the remedy phase in May 2000. The goals of the remedy phase are to undo the effects of the illegal acts, prevent recurrence of those acts, and be minimally intrusive to the company, if possible. There are generally two ways to accomplish this: structural changes (reorganization or separation of companies) and conduct changes (imposing rules). The judge could choose either or both. The decision was reached to restructure Microsoft such that the operating system (Windows) would be handled by one company and everything else by another. Furthermore, in conduct changes, Microsoft could not place limits on contracts; could not retaliate against companies for competing in other markets (such as, for example, word processing); must allow PC manufacturers to customize the operating systems on the machines they sell; must document their APIs and protocols; and cannot tie the OS and products together without providing a way to remove them.

Microsoft has appealed the case. At the time of this writing it is not clear whether the appeal will be heard in the U.S. Court of Appeals or by the U.S. Supreme Court. The remedies are stayed, or on hold, until the resolution of the appeals or until a settlement of some kind is reached between Microsoft and the U.S. government. Once the case is truly over, Felten's slides will be available on the USENIX Web site.

FREENIX SESSION: SOCKETS

Summarized by Bob Gray

PROTOCOL INDEPENDENCE USING THE SOCKETS API

Craig Metz, University of Virginia

Craig Metz convinced the audience that in an all-IP world, our network programming has become inflexible and uni-protocol. Even though the Berkeley Sockets were designed as a protocol-independent API, other parts of the API (like the name-service functions) grew up as protocol-dependent. So, some of the APIs are not protocol-independent, which in turn encourages code not to be either. For now, IPv4 is ubiquitous, but as its 32-bit address space runs out, we will find IPv6 more and more compelling. Therefore, it would be prudent to pay attention to the portability issues and even check and retrofit some of our existing network programs.

Metz pointed out multiple problems:

- Hard coded constants in programs,
- Storage limitations and assumptions,
- GUIs that assume four three-digit fields as an address.

For example, many programs hard-code the protocol family as AF_INET, which prevents protocols other than IP from being used. We should not assume a network address will fit in a u_long. And using struct sockaddr_in sin limits programs because it doesn't allocate enough storage for protocols such as IPv6.

Metz recommends using the new POSIX p1003.1g interfaces. For example, getaddrinfo performs the functionality of gethostbyname(3) and getservbyname(3), in a more sophisticated manner.

The new interfaces will take time to be universally deployed; however, they are currently available in at least the following environments: AIX, BSD/OS, FreeBSD, Linux, OpenBSD, NetBSD, Solaris, and Tru64 UNIX. They are expected to be available soon with IRIX and HP-UX.

Scalable Network I/O in Linux

Niels Provos, University of Michigan; Chuck Lever, Sun-Netscape Alliance

Graduate students Niels Provos and Chuck Lever have observed and addressed bottlenecks in Linux where many high-latency, low-bandwidth connections are simultaneously present on a Web-server box. The problem is that the stock kernel data structures and algorithms don't scale well in the presence of thousands of rapidly formed HTTP connections. The file-descriptor selection code needed work. The problem is event notification. It takes a long time for the kernel to find which connections are ready for I/O.

Niels discussed two solutions to remove the inefficiency: the POSIX RT signals API and an optimized poll() along the lines of Banga's declare_interest() interface. He convinced the audience that both mechanisms provide huge improvements for HTTP response time when 251 or 501 idle or inactive connections are present in the background. However, his optimization using /dev/poll yielded the overall best response times.

To achieve high performance, Neils made the following changes to poll():

- Maintain state information in the kernel so that every poll system call doesn't have to retransmit it.
- Allow device drivers to post completion events to poll().
- Eliminate the result copying when poll returns to the user.

The /dev/poll device allows a process to add, modify, and remove "interests" from an interest set. This streamlines poll() calls because only relevant information is passed at system call time. Further, when poll() returns, the application immediately has access to the ready descriptors.

The software enhancements to poll() are freely available.

Accept() Scalability in Linux

Stephen P. Molloy, University of Michigan; Chuck Lever, Sun-Netscape Alliance

Stephen Molloy described the thundering hard problem associated with Linux implementations of the accept() system call. When multiple threads call accept() on the same TCP socket to wait for incoming TCP connections, they are placed into a wait queue. The problem is when a TCP connection is accepted, all of the threads are awakened – even though all but one will immediately need to go back to sleep. As the number of threads goes from a few dozen to hundreds, the kernel begins severe thrashing.

One proposed solution is called Task Exclusive. The idea is to add a flag to the thread state variable, change the handling of wait queues, and connect into a standard, newly added wait-queue mechanism.

Another solution is called Wake One, which adds new calls to complement wake_up() and wake_up_interruptible(). The new functions just wake up one thread when a connection becomes ready.

Molloy presented micro-benchmark data showing huge improvements in "settle time" for both the Task Exclusive and Wake One solutions over the stock kernel.

He also used a Macro-benchmark,-SpecWeb99, to demonstrate the effectiveness of both solutions – over 50% more connections.

The Task Exclusion solution has been incorporated into the Linux kernel. The code is also available at the Linux Scalability Project's home page: *<http://www.citi.umich.edu/projects/linux-scalability/>*.

SESSION: TOOLS

Summarized by Doug Fales

Outwit: UNIX Tool-Based Programming Meets the Windows World

Diomidis D. Spinellis, University of the Aegean

Windows is fundamentally an environment of mouse clicks and pixel output. Very little of this GUI-based OS is accessible to text-based programs. It is with this shortcoming in mind that Diomidis Spinellis developed Outwit. Outwit provides a number of text-based tools for Windows to work together with UNIX-based tools. Specifically, Spinellis' tools provide mechanisms for interaction with the clipboard, the registry, the ODBC database interface, document properties, and shell links (shortcuts). The presentation included several convincing examples of the time-saving advantages of such an environment.

One example used the winreg command (the interface to the registry) to change the location of the user's home directory from drive C: to drive D:, with a little a help from pipelines and the Win32 version of sed:

```
winreg HKEY_CURRENT_USER |
sed -n 's/C:\\home/D:\\home/gp' |
winreg
```

The winclip tool provides shell-based clipboard access. For example, to copy data from standard input to the Windows clipboard:

```
ls -l | winclip -c
```

and to paste Windows clipboard data to standard output:

```
winclip -p | wc -w
```

Spinellis would be interested in adding functionality to Outwit for new features of Windows 2000, and in providing support for Unicode. The Outwit tools are available at <*http://softlab.icsd.aegean.gr/~dspin/sw/outwit*>.

PLUMBING AND OTHER UTILITIES

Rob Pike, Bell Labs

Plumbing is a Plan9 solution for interprocess communication and message passing between user applications. The general idea is to remove some of the burden on the user of constantly shuffling data from program to program. A common example of this occurs during compilation and debugging, when a compiler generates error messages detailing file location and line number. Plumbing allows the user to access the error in an editor in one click.

Thanks to the pattern-matching language at the core of the plumber's design, it is a far more powerful mechanism than filename-extension associations such as those in Windows. While the goal is for the plumber's default actions to be the most desirable actions, it is highly customizable through a configuration file that defines rules in a pattern-action format.

The actual message passing is relatively trivial because the plumber is a fileserver; messages are written to a file on the server, which then takes appropriate action based on the defined rules. The rules are actually quite flexible and powerful in that they provide a means to interpret the context of messages.

The interface for plumbing is simple and designed to minimize keystrokes and button clicks. The applications themselves do remarkably little work; almost everything is handled within the plumber itself.

One very good application of this sort of automation is in dealing with file formats that might require transformation before viewing, such as an attached Microsoft Word document. With a couple of pattern-matching rules to set up the variables, this rule takes a Word document, converts it to text, and sends it to the editor:

```
plumb start doc2txt $data | \
plumb -i -d edit \
-a action=showdata \
-a filename=$0
```

This one-click approach to conversion and viewing is a slick example of the advantages of this system. Details and more examples may be found in Pike's paper.

INTEGRATING A COMMAND SHELL INTO A WEB BROWSER

Robert C. Miller and Brad A. Myers, Carnegie Mellon University

Rob Miller demonstrated enhancements to his existing Web browser (LAPIS, see *http://www.usenix.org/events/usenix01/cfp/miller/miller_html/usenix99.html*>). The extension was an integration of browser and command shell. At first glance, this might look like another attempt to unnecessarily GUI-ify a classic typescript tool. However, this project demonstrated some very useful and innovative ways of using a Web browser.

Miller's browser provides a circular redirection of the standard input and output, so that commands are executed as if in one long pipeline. The benefits of such a mechanism in a Web browser are not immediately apparent until the full potential of conventions like the "back" button are realized.

In addition, the browser conveniently separates the standard error and output streams when displaying command output. Some features of the browser are: an embedded pattern-matching language to extract data from a Web page, a command window that can execute traditional shell commands and Web-specific commands, and the ability to automate browsing.

The demonstration showed how LAPIS could be used to visit and print (or save) each page in a document that is strung out over several links. Another interesting application was automating the use of forms. Miller used the pattern-matching language to extract the ISBN of a book from an Amazon.com Web site, and then fed this into a script that had been constructed by LAPIS to consult the form-based Web pages of CMU's library lookup service.

Judging by the demonstration, LAPIS seemed a well-designed, easy to use interface. Furthermore, it does something to alleviate the pain of endless clicking and banner-ad watching that is associated with browsing the Web these days. The browser and its Java source are available at <*http://www.cs.cmu.edu/~rcm/lapis*>.

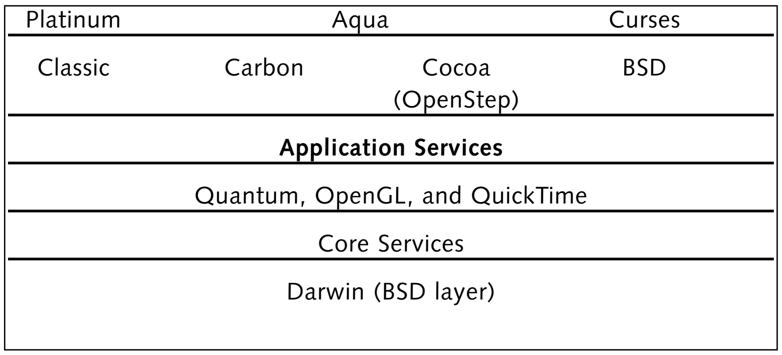
INVITED TALK

Challenges in Integrating the Mac OS and BSD Environments

Wilfredo Sanchez, Apple Computer

Summarized by Josh Simon

Fred Sanchez discussed some of the challenges in integrating the MacOS and BSD environments to produce MacOS X. Historically, the Mac was designed to provide an excellent user interface ("the best possible user experience") with tight hardware integration and a single user. In contrast UNIX was designed to solve engineering problems, using open source (for differing values of "open"), running on shared multi-user computers and with administrative overhead. There are positives and negatives with both approaches. MacOS X is based on the Mach 3.0 kernel and attempts to take the best from both worlds. A picture may help explain how all this hangs together:



Sanchez next talked about four problem areas in the integration: filesystems, files, multiple users, and backwards compatibility. Case sensitivity was not much of an issue; conflicts are rare and most substitutions are trivial. MacOS uses colon as the path separator; UNIX uses the slash. Path names change depending on whether you talk through the Carbon and Classic interfaces (colon, :) or the Cocoa and BSD interfaces (slash, /). Filename translation is also required, since it is possible for a slash to be present in a MacOS file name. File IDs are a persistent file handle that follows a file in MacOS, providing for robust alias management. However, this is not implemented in filesystems other than HFS+, so the Carbon interface provides for nonpersistent file IDs. Hard links are not supported in HFS+, but it fakes it, providing the equivalent behavior to the UFS hard link. Complex files – specifically, the MacOS data and resource forks – are in the Mac filesystems (HFS+, UFS, and NFS v4) but not the UNIX filesystems (UFS and NFS v3). The possible solutions to this problem include using AppleDouble MIME encoding, which would be good for commands like `cp` and `tar` but bad for commands using `mmap()`, or using two distinct files, which makes renaming and creating files tough, overloads the name space, and confuses `cp` and `tar`. The solution they chose was to hide both the data and resource forks underneath the filename (for example, filename/data and filename/resource, looking like a directory entry but not a directory) and have the `open()` system call return the data fork only. This lets editors and most commands (except archiving commands, like `cp` and `tar`, and `mv` across filesystem boundaries) have the expected behavior. Another filesystem problem is permissions (which exist in HFS+ and MacOS X but not in MacOS 9's HFS). The solution here is to base default permissions on the directory modes.

The second problem area is files. Special characters were allowed in MacOS filenames (including space, backslash, and the forward slash). Filename translation works around most of these problems, though users have to understand that "I/O stuff" in MacOS is the same as "I:O_stuff" in UNIX on the same machine. Also, to help reduce problems in directory permissions and handling they chose to follow NeXT's approach and treat a directory as a bundle, reducing the need for complex files and simplifying software installations, allowing drag-and-drop to install new software.

The third problem area involves multiple users. MacOS historically thought of itself as having only a single user and focused on ease of use. This lets the Mac user perform operations like setting the clock, reading any file, installing software, moving stuff around, and so on. Currently MacOS X provides hooks for UID management (such as integrating with a NetInfo or NIS or LDAP environment) and tracks known (UNIX-like) and unknown disks, disabling commands like `chown` and `chgrp` on unknown disks.

The fourth and final problem area Sanchez discussed was compatibility with legacy software and hardware. Legacy software has to "just work," and the API and toolkit cannot change, so previous binaries must continue to work unchanged. The Classic interface provides this compatibility mode. Classic is effectively a MacOS X application that runs MacOS 9 in a sandbox. This causes some disk-access problems, depending on the level (application, filesystem, disk driver, or SCSI driver). The closed architecture of the hardware is very abstracted, which helps move up the stack from low-level to the high-level application without breaking anything.

Questions focused on security, the desire to have a root account, and the terminal window or shell. The X11 windowing system can be run on MacOS X, though Apple will not be providing it. Software ports are available from <*http://www.stepwise.com/*>. Additional details can be found at <*http://www.mit.edu/people/wsanchez/papers/USENIX_2000/*>, <*http://www.apple.com/macosx/*>, and <*http://www.apple.com/darwin/*>.

FREENIX SESSION: NETWORK PUBLISHING

Summarized by Matt Grapenthien

The growth and evolution of the Internet has caused some of its limitations to become more acute. The presenters in this session attempt to address some of these problems with more dynamic and generally more useful systems and protocols than those currently in wide use.

Permanent Web Publishing

David S. H. Rosenthal, Sun Microsystems Laboratories; Vicky Reich, Stanford University Libraries

David Rosenthal presented LOCKSS (Lots of Copies Keep Stuff Safe), a system to preserve access to scientific journals in electronic form. Unlike normal systems, LOCKSS has far more replicas than necessary just to survive the anticipated failures. Exploiting the surplus of replicas, LOCKSS allows much looser coordination among them.

The Globe Distribution Network

A. Bakker, E. Amade, and G. Ballintijn, Vrije Universiteit Amsterdam; I. Kuz, Delft University of Technology; P. Verkaik, I. van der Wijk, M. van Steen, and A. S. Tanenbaum, Vrije Universiteit Amsterdam

The Globe Distribution System, presented by Arno Bakker, attempts to address the problem of distribution to a worldwide audience through selective, per-document replication, as opposed to the "all-or-none" replication policies currently in use. Though still in an early stage, the project's goal of a "better" WWW/FTP seems promising.

Open Information Pools

Johan Pouwelse, Delft University of Technology

Johan Pouwelse presented a method to allow modification and extension of existing Web pages by allowing public write access to collections of WWW-based databases. Open Information Pools further address the problem of quickly evaluating huge amounts of content, through an open rating and moderation system. Experiments on similar, already-existing systems seem to prove these concepts very valuable.

SESSION: KERNEL STRUCTURES

Summarized by Josh Kelley

Operating System Support for Multi-User, Remote, Graphical Interaction

Alexander Ya-li Wong and Margo I. Seltzer, Harvard University

An increasing number of services are being provided over the network instead of locally. Examples include filesystems (NFS), storage (Fibre Channel), memory (Distributed Shared Memory), and interfaces (thin clients). Of these, thin clients are often neglected.

The key characteristics of thin-client service are that it is interactive, multi-user, graphical, and remote. One of the most important features of thin-client service is low latency. This presentation compared two operating systems, Windows NT 4.0 Terminal Server Edition, and Linux 2.0.36 running the X Window System, and examined how well they provide these characteristics.

The first two questions regarding thin-client service are processor management and memory management. The OS's goal should be to prevent the user from experiencing any perceptible latency. To minimize latency, the OS should insure that interactive performance is background-load-independent and should swap out pages belonging to interactive processes last. Although NT offers special support for scheduling interactive threads, experiments showed Linux to be much better, both for processor scheduling and for low latencies while swapping pages in from disk. Open questions here include the best time slice to use for interactive processes and how the Linux kernel can identify interactive threads (since interactivity is determined in user space) for special treatment.

A third question regarding thin-client service is network load. In experiments, NT's Remote Display Protocol presented a much lower network load, with a larger message size, than did the X or LBX protocols used by Linux and X Windows. (This may be due partially to poorly coded X applications.) RDP's use of a client-side bitmap cache allowed it to have virtually no load in the specific area of animation (as long as the cache was not overloaded). These experiments point out the importance of a client-side cache.

Techniques for the Design of Java Operating Systems

Godmar Back, Patrick Tullmann, Leigh Stoller, Wilson C. Hsieh, and Jay Lepreau, University of Utah

A Java OS is an execution environment for Java bytecode that provides standard operating-system functionality: separation and protection, resource management, and interapplication communication. It may run on a traditional OS or it may be embedded in an application. The purpose of a Java OS is to support executing multiple Java applications.

There are several options for arranging the operating system, Java Virtual Machine, and Java applications. One approach is physical separation: one OS per one JVM per one app. Such an approach is expensive, prevents embedding of an OS within an application, and makes communication difficult. A second approach is separate JVM processes running under one OS. This approach has inefficient resource use, requires a underlying OS, and makes for difficult communication and no embedding within outside applications. A third approach is an ad hoc layer that supports running multiple applications within one JVM. Examples of this approach include an applet context or a servlet engine. However, this approach offers insufficient separation between the processes, with no resource control and unsafe termination of one runaway process. The fourth, and best, approach is to provide support for processes within the JVM, turning the JVM into a Java OS. The Java OS can be designed to replace the base OS as well.

There are several design decisions to be made for a Java OS. One example is the area of shared memory management. Issues here include the precision of accounting, the ability to reclaim shared objects, and the need for full reclamation of memory upon process termination. Java's automatic garbage collection complicates these issues. Solutions to the question of shared memory management include copying (simple but slow), indirect sharing via revocable proxies, direct sharing via a dedicated shared heap, and a hybrid approach of both direct and indirect. The J-Kernel, from Cornell University, and Alta and K0 (which later became KaffeOS), both from the University of Utah, all offer different solutions to this question and illustrate the general tradeoffs between separation, resource management, and communication.

Signaled Receiver Processing

José Brustoloni, Eran Gabber, Abraham Silberschatz, and Amit Singh, Lucent Technologies-Bell Laboratories

This session presented signaled receiver processing, an alternative to the BSD's traditional IP packet receiver processing. Since implementations of and derivatives of BSD have appeared on a variety of platforms, BSD receiver processing is a part of many operating systems, although it has several disadvantages. Protocol processing of received packets in BSD is interrupt-driven. This results in scheduling anomalies; CPU time spent processing packets is charged to the currently running process or is not charged at all. Therefore, no quality of service (QoS) guarantees are possible. A second problem is receive livelock, in which the system spends all of its time processing incoming packets, even when no buffer space is available to store these packets.

One alternative to BSD receiver processing is lazy receiver processing (LRP). To prevent receive livelock, LRP uses earlier demultiplexing to detect full receive buffers as soon as possible and drop packets accordingly. UDP packets are processed synchronously, at the receive call. TCP packets are processed asynchronously, via an extra kernel thread per process or via a system-wide process that uses resource containers. (Resource containers are an abstraction used to separate resource principal and protection domain. Resources used by kernel-level code can be charged out to user processes.) This processing of UDP and TCP packets allows LRP to avoid the BSD receiver processing's scheduling anomalies.

There are several disadvantages with LRP. First, it does not work on systems that do not implement kernel threads or resource containers. Second, under LRP, TCP is always asynchronous and shares resources equally with the application. Third, LRP is designed for hosts, not gateways. Its early demultiplexing is too simplistic for gateways, and time-sharing scheduling is inadequate for gateways. There are open questions with LRP regarding how well it would work with realtime schedulers and proportional-share schedulers.

Signaled receiver processing (SRP) is presented as an alternative method that avoids these problems with LRP. When a packet arrives, the OS signals the receiving application. By default, the packet is processed asynchronously, although the application may instead choose to catch, block, or ignore the packet, in order to defer processing until the next receive call. SRP processes incoming packets in several stages; only the actual hardware input is handled at the interrupt level. Processing is handled via a multi-stage early demultiplexer, or MED, and is transferred from one stage to another via the next stage submit (NSS) function. The NSS function signals the application by sending it a SIGUIQ (unprocessed interrupt queue).

Performance tests show that throughput, CPU utilization, and round trip times are practically the same for SRP under Eclipse/BSD as they are for BSD receiver processing under FreeBSD. Tests also show that SRP successfully prevents receive livelock. It is easily portable, allows for flexible scheduling and use with gateways, and still allows for QoS guarantees.

INVITED TALK

The Convergence of Networking and Storage: Will it be SAN or NAS?

Rod Van Meter, Network Alchemy

Summarized by Josh Simon

The goal of this talk was to provide models for thinking about SANs and NASs. Network-attached storage (NAS) is like NFS on the LAN; storage area networks (SAN) are like a bunch of Fibre Channel–attached disks.

There are several patterns of data sharing, such as one-to-many users, one-to-many locations, time slices, and fault tolerance; activities, such as read only, read-write, multiple simultaneous reads, and multiple simultaneous writes; and multiple ranges, of machines, CPUs, LAN versus WAN, and known versus unknown clients.

When sharing data over the network, how should you think about it? There are 19 principles that Levy and Silverschatz came up with that describe the file. These include the naming scheme, component unit, user mobility, availability, scalability, networking, performance, and security. There is Garth

Gibson's taxonomy of four cases: server-attached disks, like a Solaris machine; server integrated disks, like a Network Appliance machine; netSCSI, or SCSI disks shared across many hosts with one "trusted" host to do the writes, and network-attached secure devices (NASD). Over time, devices are evolving to become more and more network-attached, smarter, and programmable.

Van Meter went into several areas in more detail. Access models can be application-specific (like databases or HTTP), file-by-file (like most Unix file systems), logical blocks (like SCSI or IDE disks), or object-based (like NASD). Connections can be over any sort of transport, including Ethernet, HiPPI, Fibre Channel, ATM, SCSI, and more. Each connection model is at the physical and link layers and assumes there is a transport layer (such as TCP/IP), though other transport protocols are possible (like ST or XTP or UMTP). The issues of concurrency (are locks mandatory or advisory, is management centralized or distributed?), security (authorization and authentication, data integrity, privacy, and nonrepudiation), and network ("it doesn't matter" versus "it's all that matters") all need to be considered.

Given all those issues, there are three major classes of solutions today. The first is a distributed file system (DFS), also known as NAS. This model is a lot of computers and lots of data; examples include NFS v2, AFS, Sprite, CIFS, and XFS. The bottleneck with these systems is the file manager or object store; drawbacks include the nonprogrammability of these devices and the fact that they are OS-specific and have redundant functionality (performing the same steps different times in different layers).

The second class of solution is storage area networks (SAN). These tend to have few computers and lots of data and tend to be performance-critical. These are usually contained in a single server or machine room; the machines tend to have separate data and control networks. These devices' drawbacks are that they are neither programmable nor smart, they're too new to work well, they provide poor support for heterogeneity, and the scalability is questionable. However, there is a very low error rate and the application layer can perform data recovery. Examples of SANs include VAX clusters, NT clusters, CXFS from SGI, GFS, and SANergy.

The third solution class is NASD, developed at CMU. The devices themselves are more intelligent and perform their own file management. Clients have an NFS-like access model; disk drives enforce (but do not define) security policies. The problems with NASD is that it's too new to have reliable details, more invention is necessary, there are some OS dependencies, and some added functionality may be duplicated in different layers. Which solution is right for you? That depends on your organization's needs and priorities.

FREENIX SESSION: X11 AND USER INTERFACES

Summarized by Gustavo Vegas

The GNOME Canvas: A Generic Engine for Structured Graphics

Federico Mena-Quintero, Helix Code, Inc.; Raph Levien, Code Art Studio

The GNOME Canvas is a generic high-level engine for structured graphics. A canvas is a window in which things can be drawn. It contains a collection of graphical items such as lines, polygons, ellipses, smooth curves, and text. Graphics on this canvas are deemed to be structured because you can place these geometric shapes in the canvas and later on access the objects to change their attributes, such as position, color, and size. The canvas is in charge of all redrawing operations.

The GNOME canvas has an open interface that permits applications that use the canvas to create their own custom item types. Thus, the canvas can work as a generic display engine for all kinds of applications. The GNOME canvas items are GTK+ objects derived from an abstract class (GnomeCanvasItem), that gives the methods for objects to be implemented. Using the GTK+ object system provides several advantages, such as the possibility of associating arbitrary data items to canvas items.

The GNOME canvas also uses the Libart library for its external imaging model in antialias mode. Libart is a library that provides a superset of the PostScript imaging model, and it provides support for antialiasing and alpha transparency. The end result is that graphics' contours are smoothed out to eliminate jagged edges.

Several applications that are currently distributed as part of the GNOME environment use the GNOME canvas to render graphics and other types of data. Examples of such applications are Gnumeric (the GNOME spreadsheet), GNOME-PIM (personal information manager), and Evolution (the next-generation mail and groupware program for GNOME).

For more information about the GNOME project and the GNOME canvas, see <*http://developer.gnome.org/*>.

Efficiently Scheduling X Clients

Keith Packard, SuSE Inc.

Keith Packard presented a new scheduling algorithm for X11. The technical motivation behind this project is that the original scheduling mechanism in X11 is simplistic and can potentially starve interactive applications while a graphics-intensive program runs. This program is evident when one runs programs like plaid, which generate many rendering requests that can tie up the system for long periods of time, making it unusable by other programs, as simple as an xterm.

The X server is typically a single-threaded network server that uses well-known ports to receive connections from clients for which it processes requests sent over the port. To detect pending input from clients, it uses the select(2) system call. When the set of clients with pending input has been determined, the X server starts executing the requests, starting with the smallest file descriptor. Each client uses a buffer to read some of the data from the network connection. This buffer can be resized but it is typically 4KB. The requests are executed until either the buffer is exhausted of complete requests, or after ten requests. After this has taken place, the server figures out if there are any clients with pending complete requests. If this is the case, the server stops the select(2) call and goes back to process those pending requests. When all client input is exhausted, the X server calls select(2) again to await for more data. The problem with this algorithm is that it gives preference to more active clients. If a client generates complex requests, these requests may take up more time to be satisfied and this client will end up hogging the server. As a consequence, clients that generate fewer requests are starved in the presence of more active clients. Also, clients that do not generate a complete request during their turn will be ignored. On the positive side, when clients are busy, the server spends most of its time executing requests and wastes little time on system calls.

The design goal of the proposed solution is to provide relatively fine-grained time-based scheduling with increased priority given to applications that receive user input. Each client would be given an initial priority at connect time. Requests are executed from the client with the highest priority, and various events may change the priority of a given client. This system intends to penalize overactive clients and praise clients with little activity.

In the performance measurements that were presented a measurable change was apparent. However, the two schedulers were within only 2% of each other. This shows that the changes in the scheduler have little impact on the tool used for performance measurement. The tool used was X11perf, a widely available tool. This tool runs with very little competition from other clients, and thus most of the benefits of the new scheduler go unnoticed.

In conclusion, simple changes on the scheduler, based on real-life observations of the X server behavior, can bring some advantages over the original scheduler without impacting performance negatively. For more information and for work that has been incorporated into the 4.0 release of the X Window System from the XFree86 group, see <*http://www.xfree86.org/*>.

The AT&T AST Open Source Software Collection

Glenn S. Fowler, David G. Korn, Stephen S. North, and Kiem-Phong Vo, AT&T Laboratories-Research

David Korn presented a suite of tools that have been released to the open source community by AT&T. This tool suite includes widely known components that may or may not be directly used in graphics applications, but the fact that these have been released has a profound impact in the open source community.

These tools have been released under an AT&T license agreement. This is not GPL or LGPL, so it is important to read the license if one is going to make use of any of the software components for any purpose.

The components of this suite are deemed to be highly portable to practically any environment, given the right base. They may not necessarily be complete applications, but they can be reusable tools to produce other powerful pieces of software. Their focus when creating this software suite is reusability. They have also focused in creating software libraries that encompass core computing functions such as I/O and memory allocation and other new algorithms and data structures such as data compression and differencing and graph drawing. Thus, they created libraries like,

- Libast – Porting base library for their software tools.
- Sfio – This I/O library provides a robust interface and implements new buffering and data formatting algorithms that are more efficient than those in the standard I/O library, Stdio.
- Vmalloc – This memory-allocation library allows creation of different memory regions based on application-defined memory types (heap, shared, memory mapped, etc.) and some library-provided memory-management strategies.
- Cdt – This container data-type library provides a comprehensive set of containers under a unified interface: ordered/unordered sets/multisets, lists, stacks, and queues.
- Libexpr – This library provides run-time evaluation for simple C-styled expressions.
- Libgraph – This graph library supports attributed graphs, generalized nested subgraphs, and stream file I/O in a flexible graph data language. It is built on top of the `Cdt` library and employs disciplines for I/O, memory management, graph-object namespace management and object-update callbacks. This library is the base of the Graphviz package.

Other complete tools that have been released as part of this collection are reimplementations of programs like the KornShell language and nmake, and other new applications like: `tw`, a more powerful find and xargs; and warp, a tool that helped with Y2K testing by running a process through it and simulation a time and clock speed different from the actual system's time and clock.

For more information, please see <*http://www.research.att.com/sw/tools/*>. For the AT&T source code license agreement, please see <http://www.research.att.com/sw/license/ast-open.html>.

SESSION: WORKS IN PROGRESS

Summarized by Kevin Fu

Perl on the Itanium

Murray Nesbitt <murray@activestate.com>, ActiveState Tool Corp.

Nesbitt discussed his experiences in building Perl under Windows 2000, Linux, and Monterey on the 64-bit Intel Itanium. It was straightforward to build Perl on IA-64 Linux. Building Perl on Monterey was almost as easy; it required a standard Perl hints file to compile though. Murray's main point was that building Perl on Windows 2000 was more painful for a number of reasons such as lack of configure-script support and problems with type sizes and abstractions. In the future Murray plans to work on optimization. In short, it's fairly easy to port Perl, but it's helpful to share an office with a Perl guru.

ttf2pt1: A TTF to Adobe Type 1 Font Converter

Sergey Babkin <babkin@users.sourceforge.net>, Santa Cruz Operation

Babkin described his work on a TTF-to-Adobe Type 1 converter. The converter also attempts to clean the outlines from detects and automatically generate Type 1 hints. His program optimizes the method to make the conversion look good. It comes under the BSD license and is reasonably well modularized. He noted that this work has nothing to do with SCO and is simply his personal hobby. For more information, see <*http://ttf2pt1.sourceforge.net/*>.

lodd: A Pipelining and In-Pipe Data Manipulation Tool

Joseph Pingenot <jap3003@ksu.edu>, Kansas State University

Pingenot, an undergraduate at KSU, talked about a tool to combine multiple channels of input and output. The lodd utility hopes to perform tasks such as acting as a logical dd and mixing three pipes together. lodd 1.x has stream-pipe support and can mix multiple pipelines together, manipulate data at the block level, perform bitwise logic, and divide pipelines. lodd is dd-compatible. Some of the interesting issues in creating lodd include dealing with blocking I/O, back-streams, block sizes, shell limitations, and deciding what to do if a pipe closes. Visit <*http://www.phys.ksu.edu/~trelane/lodd*> for more information.

Q: Are functions extensible? A: Not in the preliminary version. Hopefully in the future.

Q: How does one use lodd within a shell? A: I am now looking at ways to implement a shell syntax. Suggestions are welcome.

vstack: Easily Catch Some Buffer Overrun Attacks

Craig Metz <cmetz@inner.net>, University of Virginia

Buffer overruns typically work by overwriting a function's return address with a value of the adversary's choice. In this way, an adversary can change the flow of control to execute, for example, a root shell. Metz described a simple approach that prevents many commonly exploited overruns.

vstack verifies that function return addresses do not change. It does so by keeping a separate virtual stack of return addresses and frame pointers. On return from a function, a program verifies that the return address and frame pointer on the execution stack matches that on the virtual stack. If not, the program jumps to a fault handler. This does not break standard calling convention and requires changes only to the caller convention in a compiler.

A sample Perl implementation exists for the x86. It edits assembly code to insert the checks and management of the virtual stack. The code will appear soon under a BSD-style license. The performance loss is minimal. In the future, Metz plans to have more sophisticated choices in what to do after detecting an overrun. vstack does not catch every overrun attack, but it can catch the vast majority.

Q: Instead of verifying the return address matches, why not simply use the return address on the virtual stack? A: There might be other corrupted data. In special cases it might be OK to use the virtual stack directly, but not in general.

Q: What prevents overwriting the virtual stack itself? A: It is elsewhere in memory. If you can overwrite an extent of 2^{32} space, then you can overwrite everything anyway.

Q: Does longjmp() or other functions that unwind the real stack confuse vstack? A: Probably. (Offline Metz explained that this is mostly solvable by using wrappers around longjmp() and related functions. It won't catch every case, but it will catch most of them.)

Q: Can I overwrite data on the stack? A: Yes, but vstack will detect changes made to return addresses.

kq: Kernel Queues in FreeBSD

John-Mark Gurney <jmg@freebsd.org>, FreeBSD

Kernel queues are a stateful method of event notification. Instead of passing which events to monitor each time as is done with select(2) and poll(2), the program tells the kernel which events need notification. kq supports event monitors (filters) for file descriptors, processes, signals, asynchronous I/O, and VNODEs. State is allocated in kernel memory. kq pays attention to what file descriptors are ready for reading and writing.

John-Mark described l0pht's watch program modified to use kq. The watch program looks for temporary files created in /tmp. Before using kq, the watch program consumed a lot of CPU time by polling directory entries in /tmp over and over. With kq, you get a notification when the /tmp directory changes. In this manner, the watch program does not needlessly scan an unmodified directory.

Efforts are underway to use kq in the Squid HTTP proxy cache for asynchronous I/O and in ircd to reduce the server load. Visit *<http://people.freebsd.org/~jmg/kq.html>* for more information.

Q: How does this compare to /dev/poll on Solaris 8? A: I haven't looked at /dev/poll; it's hard to say. However, my system is extremely lightweight.

Q: If a given source sends multiple signals, will it cause a series of kernel memory allocations before the read occurs? A: The memory is actually allocated when you register.

Q: Can you unify with other kernel name spaces? A: Currently you are limited to a filesystem. However, pretty much any kernel object can be associated.

Autonomous Service Composition on the Web

Laurence Melloul <melloul@stanford.edu>, Stanford University

The goal of this service it to allow dynamic specification of composition requests. The advantages include a cost-effective development cycle, better fault tolerance, and high availability. Melloul chose the Web as the medium because it has autonomous services, uses a public infrastructure, is common, is simple, and speaks a language independent protocol (HTTP). The two main issues are detection of service interface changes and verification of semantic compatibility between service parameters. The key is to build on the ontology by web users.

The Hummingbird File System

Liddy Shriver <shriver@research.bell-labs.com>, Bell Labs, Lucent Technologies

The Hummingbird File System caters to workloads of a caching Web proxy. On UNIX machines, server software such as Apache or Squid typically runs on a derivative of the 4.2BSD Fast File System. FFS was not designed with the workload of a proxy server in mind. In particular a Web-proxy workload has high temporal locality, relaxed persistence, and a read/write ratio different from most workloads. FFS also includes features which a proxy server does not require.

The Hummingbird File System takes advantage of Web-proxy server properties such as whole file access, small files on average, and repeatable reference locality sets. It also co-locates the storage of an HTML Web page with its embedded GIFs. Performance measurements show that under a sample Web-proxy workload, Hummingbird supports throughput five to ten times greater than that of XFS and EFS (SGI) and six to ten times greater than that of FFS mounted asynchronously (FreeBSD).

Future work includes persistence of data. At the moment, Hummingbird does not worry about persistence because the data can be regenerated from origin servers. Visit <http://www.bell-labs.com/project/hummingbird/> for more information.

Q: Does Hummingbird support for HTTP reads for specific byte ranges? A: Not yet.

Q: When does Hummingbird write to disk? A: During idle time and when memory is filled and space is needed.

The Self-Certifying File System

Kevin Fu <fubob@mit.edu>, MIT Lab for Computer Science

The Self-Certifying File System is a secure, global filesystem with completely decentralized control. SFS lets you access your files from anywhere and share them with anyone, anywhere. Anyone can set up an SFS server, and any user can access any server from any client. SFS lets you share files across administrative realms without involving administrators or certificate authorities.

SFS is a secure network filesystem in the sense that it provides confidentiality and integrity of the Remote Procedure Calls (RPCs) going over the wire. SFS runs at user level and uses NFSv3 for portability. It performs between TCP and UDP NFS on FreeBSD and is in day-to-day use for Fu's research group.

Server public keys are made explicit in pathnames. The pathname to a remote file includes a "HostID" that consists of a cryptographic hash of a server's public key and hostname. At a high level, the HostID is essentially equivalent to a public key. Using this convention, we can easily implement certificate authorities with symbolic links. Then a certificate authority can chain trust with symbolic links. A system of user agents and authenticated lists of trusted HostId takes care of most of the HostIds.

There is also a read-only dialect suitable for highly replicated, public, read-only data (e.g., software-distribution or certificate authorities). In this scheme, an administrator creates offline a signed database of a filesystem to export. Untrusted servers can replicate this database. Clients can then select any of the untrusted servers. Because the database is signed and the self-certifying path denotes the corresponding public key, the client can verify that the data is authentic. Measurements show that a read-only server can handle many times the workload of a read-write SFS server and that server-side authentication is more than an order of magnitude faster than that of SSL.

SFS is free software. The SFS developers have used it on OpenBSD, FreeBSD, Solaris, OSF/1, and a patched Linux kernel. Download the software from <*http://www.fs.net/*>.

Q: Why not store server public keys in a file? Is the self-certification just a hack? A: We believe the symlink approach is more elegant. And c'mon! It's a cool hack.

Q: Have you thought about committing this to the FreeBSD tree? You should make sure that your software stays maintained by contacting someone in charge of distributions for each operating system. A: You are certainly welcome to include SFS in your operating system. We can try to locate the appropriate contacts for each operating system, but you are welcome to contact the SFS developers too. Email <*sfs-dev@pdos.lcs.mit.edu*>.

Q: Do you rely on DNS for security? A: No. We only use DNS as a hint to locate a server. If a fake SFS server responds to a request, the client will detect the fake because the fake server's private key will not correspond to the public key described in the self-certifying path. You could receive notification of such failures via an agent. In our system, the worst an adversary can do is deny service.

NFS Version 4 Open Systems Project

Andy Adamson <andros@umich.edu>, CITI, University of Michigan

Adamson discussed some of the interesting features of NFS version 4. The CITI group at the University of Michigan received funding from Sun Microsystems to implement NFSv4 on Linux and OpenBSD.

NFS version 4 has compound RPCs that perform multiple operations per RPC. There is no more mountd. There is no more lockd. Locking is incorporated into the protocol and includes DOS share locks and nonblocking byte-range locks with lease-based recovery. Delegation aids in client file cache consistency. The server controls who gets delegation. Security is added to the RPC layer via GSSAPI. NFSv4 requires Kerberos5 and Lipkey PKI implementations. The security mechanism and QOP is negotiated between the client and server.

The code has passed all nine basic Connectathon tests under Linux. The CITI folks have just started work on the OpenBSD code. In the near future, Adamson plans to rebase to Linux 2.2.4 and finish the OpenBSD port. Source code will be available by September 1, 2000. Visit <*http://www.citi.umich.edu/projects/nfsv4/*> for more information.

Q: Is there backwards compatibility with NFS3? A: There is none.

Q: Do you expect future growth in NFS specifications? A: Ohhhh yeah.

Q: What are the terms of the license? A: Under Linux it is GPLed. Under OpenBSD it has the OpenBSD license.

Q: Does it work under IPv6? A: We'd love it to work with IPv6. Would you like to financially support us?

Alfa-1: A Simulated Computer with Educational Purpose

Alejandro Troccoli <atroccol@dc.uba.ar> and Sergio Zlotnik <szlotnik@dc.uba.ar>, University of Buenos Aires

The Alfa-1 project consists of software tools to simulate a processor. This helps in teaching computer architecture to undergraduate students. The GAD tool was used as a basis for developing a simulated computer, allowing students to experiment with the approach defined by the DEVS formalism. The model is based mainly on the specification of the Integer Unit of the Sparc processor.

Undergraduates implemented most of the system, which is now completely specified and implemented. In the future, the Alfa-1 staff hopes to add a GUI, perform exhaustive testing, use digital logic gates, add another level of cache memory, and promote educational use of Alfa-1. Visit <*http://www.dc.uba.ar/people/proyinv/usenix/*> for more information.

Q: What are you able to simulate? A: If we had enough computational power, we could simulate everything.

Q: Are there triggers, tracepoints, GDB for this? A: This is plain C code. You can debug the simulator using any standard tool.

Tellme Studio

Jeff Kellem <composer@tellme.com>, Tellme Networks

Tellme Studio offers a free service for developing and testing voice XML applications. All you need is knowledge of VoiceXML and JavaScript (if you choose to use JavaScript). You write your VoiceXML code, put it up on a Web server somewhere, log in to Tellme Studio (<*http://studio.tellme.com/*>), and give the URL pointing to your code. You are then given an 800 number to call to immediately test out the application. Tellme Studio includes VoiceXML documentation, code and grammar examples, and a community for sharing ideas.

Passwords Found on a Wireless Network

Dug Song <dugsong@monkey.org>, CITI, UMich

Receiving a standing ovation and giving by far the most entertaining talk at the conference, Dug Song from the CITI group at the University of Michigan gave a "brief report of what he found in the air." He further described tools he created to demonstrate the insecurity of his network. In the process, the audience convinced him to give a live demonstration on how easy it is to collect passwords and shadow a user's Web surfing.

Song's second slide included slightly sanitized sniffer logs. Among the finds were cleartext root logins via Telnet and colorful passwords such as "hello dug song, do I smell." While explaining an ebay.com URL, Dug reasoned that, "I don't know if Matt Blaze is here, but it sure looks like he is."

On to more serious stuff, Dug explained the rationale behind his seemingly malicious behavior. He views himself not as a bad guy, but as someone promoting "security through public humiliation." On that note, he introduced the mother of all password sniffers and a few penetration testing tools. His penetration tools include:

arpredirect – This tool is extremely effective for sniffing on switched Ethernet. It does so by poisoning ARP. It politely restores the ARP mappings when finished.
macof – This is a C port of a tool to flood a network with random MAC addresses. It causes some switches to fail open in repeating mode. This effectively turns the switch into a hub for the purposes of sniffing. Dug commented, "Switch becomes hub, sniffing is good."
tcpkill – A evil tool to selectively kill connections. However, Dug uses it to remotely initialize connection state.
tcpnice – This selectively slows down traffic by using ICMP quenches and shrinking TCP window sizes. This is useful for sniffers that work better on slower network traffic. It's also useful against things like Napster.
dsniff – This sniffer decodes 30 major protocols from Telnet to Meeting Maker. The HTTP module recognizes password URL schemes for many e-commerce sites (e.g., Web mail sites, eBay, etc). dsniff uses a magic(5)-style automatic protocol detection. For example, running Telnet on port 3000 will not fool dsniff because it can determine protocol by analyzing the traffic content.
filesnarf – Sucks down cleartext NFS2 and NFS3 traffic. Very useful against files such as .XAuthorithy and .ssh/identity. This is Song's "motivation" for developing NFSv4.
mailsnar – A fast and easy way to violate the Electronic Communications Privacy Act of 1986 (18 USC 2701-2711), be careful. This snarfs cleartext mail and outputs the contents into a convenient format suitable for offline browsing with your favorite mail reader.
urlsnarf – Same idea as mailsnarf but for URLs.
webspy – Very sinister. It allows you to watch someone's Web surfing in real-time. Dug demonstrated the tool by shadowing the Web surfing of an audience member.

Song concluded that many people incorrectly believe wireless and switched networks are immune to sniffing. He thinks that public humiliation can remind people of this misconception. Visit <*http://www.monkey.org/~dugsong/dsniff*> for more information. The slides are on <*http://www.monkey.org/~dugsong/talks/usenix00.ps*>.

Q: Should we block access to port 23 at USENIX terminal rooms? Then cleartext Telnet sessions will not happen. A: That's a technical solution to a social problem. I like my way better.

Q: The conference network ran out of DHCP leases. Can your tools help me? A: During the conference I wrote a short "dhcpfree" program to forcibly free up IP addresses. [Crowd laughs as he shows the code.]

Q: Once upon a time in a terminal room, I measured the ratio of SSH/Telnet traffic. At one point, I said loudly "look at all the interesting passwords!" All of a sudden, the ratio went up. A: Yup.

INVITED TALK

Lessons Learned about Open Source

Jim Gettys, Compaq

Summarized by Matt Grapenthien

In this interesting, informative, and thoroughly entertaining talk, Jim Gettys addressed the history of several projects and presented the most (and least) successful methodologies over a time frame of about two decades.

Beginning with a history of X, Gettys traced its peaks and valleys from "prehistory" (1983) through our present "baroque" period. When the CDE (or "cruddy desktop environment") took over the market, X development became almost nonexistent. Then, starting in about 1996, a combination of factors revitalized X. X is better today than any point in the past, and the future looks promising.

Jim Gettys

Next Gettys talked about several individual projects, tracing the Apache's market share through the past decade. Through these case studies, he noted which practices worked best (release continuously, make it easy for developers to contribute) and which didn't work at all.

Gettys achieved a rare balance of keeping the talk both very entertaining and very useful. His wit, sarcasm, and experience (20 years with OSS) made this a valuable and enjoyable session.

SESSION: RUN-TIME TOOLS AND TRICKS

Summarized by Josh Kelley

DITOOLS: APPLICATION-LEVEL SUPPORT FOR DYNAMIC EXTENSION AND FLEXIBLE COMPOSITION

Albert Serra, Nacho Navarro, and Toni Cortes, Universitat Politècnica de Catalunya

DITools is a way to modify an application or library without access to the source code. It can be used, for example, to profile a binary without instrumentation, to use another library with a program without rewriting the program, or to fix a bug in a library without recompiling it.

A process image is traditionally built of three parts: a runtime loader, a main program, and one or more libraries. The OS brings all of the needed modules into the address space, and then the runtime loader resolves references between these modules. DITools augments the loader to insert an extension backend between the main program and the libraries. The main program then calls the extension backend instead of the libraries, and the backend forwards calls to the libraries as appropriate.

DITools loads before entering the program. It offers dynamic loading support and uses binding management to interpose modules. DITools can change function bindings on a per-module basis, through rebinding, or globally, through redefinition. DITools also offers two interposition modes; it can change the linkage table to point directly to a backend's wrapper, or it can change the linkage table to point to DITools's dispatcher, which calls callback functions in the backend before and after calling the original function. To ensure correct operation, DITools transparently checks for and handles events that may affect its behavior, such as dynamic loading, process forking, and multithreading.

Performance tests show that DITools incurs a reasonable overhead on function calls. DITools complements related methods such as static binary rewriting or dynamic instrumentation based on code patching. DITools operates at a higher abstraction level and is simpler than these related methods, but it works at the function level only. It presents an application-level tool to intercept cross-module references and easily make changes. DITools is available from <*http://www.ac.upc.es/recerca/CAP/DITools*>.

PORTABLE MULTITHREADING-THE SIGNAL STACK TRICK FOR USER-SPACE THREAD CREATION

Ralf S. Engelschall, Technische Universität München (TUM)

Multithreading offers many advantages to a programmer, but finding a portable fallback approach to implement threading on UNIX platforms can be difficult, if the standardized Pthreads API is not available. Setjmp(3) and longjmp(3) are the traditional methods of transferring execution control in user-space, but they do not address the question of how to create a machine context on a particular runtime stack. The ucontext(3) API allows for user-space context creation and switching, but it is still not available across all platforms.

A solution to this problem is to use the UNIX signal-handling facilities in conjunction with setjmp(3) and longjmp(3). The process can create a machine context by setting up a signal stack using sigaltstack(2), then sending itself a signal to transfer control onto that stack. Once in that stack, the process saves the machine context there via setjmp(3), leaves the signal handler scope, and later restores the saved machine context without signal-handler scope. Then it finally enters the thread-startup routine while running on this particular stack. A much more detailed description of the algorithm is available in the author's paper or at <*http://www.engelschall.com/pw/usenix/2000/*>).

Performance tests show that thread creation using this signal stack trick is about 15 times slower than thread creation using ucontext(3), because of the signaling required. However, user-space context switching is as fast as with ucontext(3). The signal-stack trick offers an extremely portable method of implementing user-space threads without relying on assembly code or platform-specific facilities. This fallback approach is used in the GNU Portable Threads (Pth) library, available from <*http://www.gnu.org/software/pth/*>.

TRANSPARENT RUN-TIME DEFENSE AGAINST STACK-SMASHING ATTACKS

Arash Baratloo and Navjot Singh, Bell Labs Research, Lucent Technologies; Timothy Tsai, Reliable Software Technologies

Buffer overflows are one of the most common sources of security vulnerabilities. Crackers can exploit buffer overflows to achieve two dependent goals: injecting attack code and altering control flow to execute this attack flow. The basic method of exploiting a buffer overflow is the stack-smashing attack, where the attacker puts the attack code on the stack, then overwrites the current function's return address with the address of the attack code. This presentation offered two complementary defenses against stack-smashing attacks.

The first defense is libsafe, which intercepts calls to unsafe functions and replaces them with safe versions. The majority of buffer overflows result from the misuse of unsafe functions such as strcpy and fscanf. At runtime, libsafe estimates a safe upper bound on the stack buffer. It intercepts calls to these unsafe functions and replaces them with calls using this upper bound, thus containing overflows to a safe region and guaranteeing that the stack return addresses are protected. The function-interception technique is similar to that used by zlib.

The second defense, libverify, uses binary rewrites to ensure that return addresses are valid before use. It wraps each function to save the function's return address on the heap upon entry and check the return address at exit. If the return address has changed, then the process displays an error to screen and syslog and dies. libverify uses runtime instrumentation (copies the program at runtime and changes it) to insert its stack-checking code. This technique is similar to that used by StackGuard, but without recompilation of the source code.

Both libraries can be loaded for an already-compiled binary using an entry in /etc/ld.so.preload or the LD_PRELOAD environment variable. Testing showed that these libraries successfully prevented known exploits on several programs with reasonable execution time overhead. libsafe is available for Linux from *<http://www.bell-labs.com/org/11356/libsafe.html>*.

INVITED TALK

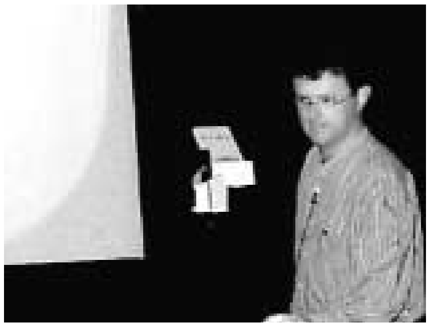
An Introduction to Quantum Computation and Communication

Rob Pike, Lucent Technologies – Bell Labs

Summarized by Doug Fales

Rob Pike's discussion of quantum computing was a very forward-looking, change-of-pace invited talk. He first reviewed some quantum mechanics to bring the audience to common ground. The always-popular polarized-light experiment helped to demonstrate the principles. He also presented the famous two-slit experiment, in which a single photon passed through two very small slits in a barrier still creates interference on the opposite side of the barrier. Pike used this as a demonstration of the Quantum Measurement Postulate because when the particle is measured to see which slit it passed through, the pattern disappears.

After the simplified (but challenging) introduction, Pike progressed into the more specific field of quantum computation. He addressed quantum-mechanical phenomena like decoherence and entanglement, as well as the implementation of quantum computational systems (qubits, quantum gates, etc.). The most promising aspects of this infant technology, those of massive parallelism and zero-energy calculation, were clarified. As examples of the power of quantum computation, Pike went over the possibilities of Shor's algorithm for factoring large primes in polynomial time, and Grover's algorithm, which searches a list in square root time.


Rob Pike

In addition to the computational side of quantum mechanics, Pike also addressed possibilities for communications, which he saw as not so distant in the future. This included a discussion of EPR pairs (a pair of entangled photons produced by electron-positron annihilation, named after Einstein, Podolsky and Rosen) and how entanglement is actually an advantage for communications.

Perhaps the most interesting point in the whole talk was the theme that information is known to be a physical quantity, restricted by a law of conservation, much like energy or mass. Furthermore, as Moore's Law continues to shrink classical computers, we will run into a physical barrier of size; as Pike put it, "we're running out of particles."

Although quantum computation is still relatively far from real application, Pike noted in closing that we cannot tell yet whether progress in this field will be linear or exponential. After all, he said, classical computers were as much of an enigma 60 years ago as quantum computation is today. The slides for this presentation are at *<http://www.usenix.org/events/usenix2000/invitedtalks/pike_html/index.html>*.

INVITED TALK

Providing Future Web Services

Andy Poggio, Sun Labs

Summarized by Josh Simon

Andy Poggio basically expanded on Bill Joy's keynote talk. The Internet has effectively begun to mimic Main Street and is beginning to provide those services that Main Street cannot, such as any time and anywhere. The six Webs are of relevance:

Near web – Monitor, keyboard, and mouse attached to a nearby system; personalized news such as multimedia and news-on-demand; and educational aspects like multimedia, interactive simulations, and so on. An example of educational uses of the near web can be found at *<http://www.planetit.com/techcenters/docs/>*.

Far web – The television or appliance with remote control, providing entertainment on demand; multiple data sources, providing a lower barrier to entry; possibly targeted advertising with product placement in on-demand movies showing Coke ads on the sides of a taxi cab to Coke drinkers but showing Pepsi drinkers a Pepsi ad in the same position.

Voice web – For use when the hand and eye are busy, like driving a car.

e-commerce web – Computer-to-computer, such as auctions (both "forward" like eBay and "reverse" like eWanted) and dynamic pricing.

Device web – Device-to-device for non-PC devices like cell phones and pagers and set-top boxes. These include agents to collect data and remote distributed processing via IP and Java.

Here web – Personal digital assets, like "my CDs" or "my MP3s" or "my DVDs"; providing on-demand access and ownership, and allowing the end-user to create her own environments.

So how do we get there? Three aspects need to be worked on. First, the network has to be enhanced. IPv6 provides more address space, better configuration management, authentication, and authorization, but adoption has been slow. Poggio predicts wired devices will win over wireless devices, both quality of service and overprovisioning will continue, optical fiber will replace or supercede electrical (copper) wiring, and the last mile to the home or the consumer will be fiber instead of ADSL or cable modems or satellites. Second, the computer-chip architecture will probably remain based on silicon for the next ten or so years. Quantum effects (see the "Quantum Computing" talk for more information) show up around 0.02 microns, so we need new approaches such as optical computing, organic computing, quantum computing, or computational fogs (virtual realities). Third, Poggio believes that the system architecture will connect three components – CPU server, storage devices, and the network – with some form of fast pipe, probably InfiniBand (a high-bandwidth, low-error, low-latency fast interconnect).

FREENIX SESSION: COOL STUFF

Summarized by Jeff Schouten

An Operating System in Java for the Lego Mindstorms RCX Microcontroller

Pekka Nikander, Helsinki University of Technology

The RCX Microcontroller is a device sold as part of a Lego set. It's designed to move a bit of Lego here or there, and making a mostly functional robot for a child to play with. As a project, a Java operating system was developed for this microcontroller by Pekka Nikander and his students at Helsinki University of Technology. The RCX consists of a Hitachi H8 microcontroller, 32K of ram, an LCD panel, an IR transceiver, and several IO devices.

Using mostly Java, a small bit of C++, and a bit of H8 assembly, there is a mostly functional OS for the RCX. When the RCX agrees to take the download (about one in three times) it runs fairly well, but gets stuck in a loop once in a while. A student is currently debugging this behavior. (That gave us a few laughs).

In all, it's an amazingly small OS, for a very limited task, but it works – it can be done.

LAP: A Little Language for OS Emulation

Donn M. Seeley, Berkeley Software Design, Inc.

LAP (Linux Application Platform), is a Linux-emulation package for BSD/OS that allows Linux applications to run under BSD/OS. By loading a shared library on a BSD system to "catch" Linux system-level calls and reroute them to the BSD kernel, a BSD user can run a Linux application.

Emulation is quite successful. A number of interesting Linux applications run via LAP and liblinux under BSD, including Adobe Acrobat Reader v4, Netscape Communicator, and WordPerfect 8. Not all functions of a Linux system are emulated, most notably the Linux clone() system call.

Overall, it seems to not eat a lot of processor, and it provides BSD users a bit of flexibility they didn't previously have.

Traffic Data Repository at the WIDE Project

Kenjiro Cho, Sony CSL; Koushirou Mitsuya, Keio University; Akira Kato, University of Tokyo

The idea behind this project is to collect statistical data on trans-Pacific backbone links on the Internet. WIDE is a Japanese research consortium that designed this data repository, with the intent of building free tools with which to build your own repository.

One problem with this type of data is privacy, another security. How do they protect private information from leaking into the repository and thus generating a possible security breach? Removal of the payload is the first step, leaving only header information to analyze. Address Scrambling is the second step, rewriting or stripping out IP addresses out of ICMP and TCP packets.

INVITED TALK

The GNOME Project

Miguel de Icaza

Summarized by Josh Kelley

Although Linux has proved itself on the server, its progress on the desktop lagged until quite recently. Three years ago, UNIX had little innovation; the last significant user interface change was X Windows. There was little code reuse and no consistent way to build desktop applications. Improvements were incremental enhancements to speed and feature lists rather than major architectural changes, and there was little direction between groups making these enhancements. The

GNOME project aims to correct these shortcomings.

The GNOME project is a unified, concerted effort to build a free desktop environment for UNIX. One major part of this effort is GNOME's component platform. GNOME is designed as a collection of components that build on top of one another; dependencies between components are encouraged, and each application or component exports its internals to others via CORBA.

This approach to writing software as a collection of small components works well with free software. Since free-software contributors tend to come and go, a set of components allows them to focus on, and contribute to, a small problem with relatively little ramp-up time.

Traditional approaches to components have several disadvantages. UNIX command-line tools may not be easy to use and can only communicate through unidirectional pipes that transfer primarily streams of textual data. Object-oriented programming has its advantages, but sharing objects among different object-oriented languages is difficult.

Bonobo is the GNOME solution for components. Bonobo is a component architecture based on CORBA and partially inspired by Microsoft's COM/ActiveX/OLE2. Bonobo provides the building blocks and the core infrastructure for writing and using components. Bonobo can be divided into two parts: the CORBA interfaces that are the contract between the providers and the users, and the implementations of those interfaces. Other implementations of these interfaces are possible. For example, KDE could choose to provide these interfaces to allow KDE and GNOME components to work together.

Since Bonobo is based on CORBA, it has all of the standard CORBA features, including language independence and support for automation and scripting. Bonobo's basic interface is `Bonobo::Unknown`, which provides two basic features: life-cycle management of an object through reference counting, and dynamic discovery of features through a standard query interface. Specific components may support any number of additional interfaces to allow full access to their functionality.

One application of Bonobo would be to provide standard interfaces to system services. Traditional UNIX, for example, stores configuration information in a variety of formats, mostly in files under the `/etc` directory, and specific instructions on how to make changes and how to put these changes into effect often differ. The goal of Bonobo (and GNOME) is to have CORBA interfaces everywhere, for every service, for the desktop, and for each application. The entire system should be scriptable. Applications should have easy access to one another's functionality rather than having to write desired functionality themselves. This provides better IPC (a more flexible alternative to `pipe` and `fork`) and better Internet protocols (applications can communicate via Bonobo rather than using ad hoc protocols).

Bonobo is used in several applications that are either in development or are currently available. These include Gnumeric (a spreadsheet), Sodipodi (a draw application), Evolution (a mail and calendar system), and Nautilus (the GNOME 2.0 file manager). Bonobo is also integrated with the rest of GNOME; the GNOME canvas, for example, allows embedding of objects of any type, and the GNOME printing architecture offers functionality similar to the canvas to provide an alternative to using PostScript for everything. GNOME 1.4, which includes Bonobo 1.0, should be released this October.

FREENIX SESSION: SHORT TOPICS

Summarized by Craig Soules

JEMACS – THE JAVA/SCHEME-BASED EMACS

Per Bothner

The first talk of the "shorts" section of Freenix was a discussion of JEmacs, the Java/Scheme-based Emacs. Described as "A next-generation Emacs based on Java," JEmacs offers all of the standard features of Emacs, as well as a number of useful new features.

The rationale behind JEmacs is that Java implementations have become increasingly faster, making it suitable to an application such as Emacs. Additionally, it offers a number of useful features, such as built-in Unicode support and multithreading. Through the use of Kawa, JEmacs also has an easy-to-use Scheme interpreter that offers features of Java, such as Swing (its GUI interface), in a simple scripting language.

In order to make the transition to JEmacs as painless as possible, JEmacs also offers full ELisp support. Although there are some slight difficulties with integrating ELisp with the multithreaded nature of Java, the author seems to have them well in hand. For more information on JEmacs see <*http://www.jemacs.net/*>. Kawa (JEmacs's Scheme implementation) also has a home page, <*http://www.gnu.org/software/kawa/*>.

A NEW RENDERING MODEL FOR X

Keith Packard, SuSE, Inc.

Keith Packard spoke on developing a new rendering model for the X Window System. The current X system was developed based upon the PostScript specification of the time, and was targeted at being a simple solution to support simple "business" applications. Today most programs don't even use many of the available features of X, relying on inefficient or unaccelerated external libraries to handle screen drawing, which is done mostly client side. By looking at the requirements of most X programs today, Packard has developed a new model for X that will offer many of its missing features.

Although many things in the current X system are lacking, there are a few things that are worth keeping, such as testable pixelization and exposed pixel values. It is important also, that all of the potential, good and bad, of the old model is left in place for use by legacy programs. The proposed solution offered here is simply to add a number of new features. These include: alpha composition, antialiasing support, a finer-grained coordinate system, more rendering primitives, and better text handling.

UBC: An Efficient Unified I/O and Memory Caching Subsystem for NetBSD

Chuck Silvers, The NetBSD Project

UBC is a unification of the I/O and memory caching systems of NetBSD, and was presented by Chuck Silvers. Unlike many other current operating systems, NetBSD had a separated page and file caching. This led to many complications within the kernel involving management between the two systems to avoid stale data and proper behavior. Additionally, the page cache has a number of useful features not available in the file-buffer cache, such as being dynamically resized.

The proposed solution to this problem is to have the page cache manage all file access. This offers a number of benefits: only one copy of the data will ever be cached, which also prevents any copying to avoid stale data; the page cache can dynamically resize itself; and cached data no longer needs to be constantly mapped in memory to remain in the cache. This is all managed through several new calls, which are used by the buffer cache to retrieve and manage pages from the virtual-memory system.

Although this new system has the potential to have increased performance, as well as reducing overall cache size, many improvements need to be made in order to make this a reality. Tests with the initial system indicate worse sequential-access performance than the current caching system. This is claimed to be due mostly to unaggressive read ahead and bad pager algorithms. In addition to the performance tuning, several other enhancements are in the future, such as soft-updates support, avoiding the need to map pages in order to do file I/O, and page loan out. The current implementation of this code will become available in the release following the 1.5 release of NetBSD.

Mbuf Issues in 4.4BSD IPv6 Support – Experiences from the KAME IPv6/IPsec Implementation

Jun-ichiro itojun Hagino, Internet Initiative Japan, Inc.

This presentation looked at the difficulties found in offering IPv6 support in a BSD environment. Although IPv6 was designed with the idea that it can easily be layered over current IPv4 implementations, its implementation on 4.4BSD proved to be more complicated than advertised.

These complications arose from a number of IPv4-specific assumptions made within the BSD kernel. The biggest of these problems arose from a change in header size in IPv6, which led to severe packet loss. In order to fix this problem, the author created a new parser for the IP layer which not only offered correct IPv6 support, but also reduced the amount of copying and `mbuf` allocation significantly. This work has now been integrated with all of the major BSD kernels available. For more information on this work, see <*http://www.kame.net/*>.

Malloc() Performance in a Multithreaded Linux Environment

Chuck Lever and David Boreham, Sun-Netscape Alliance

This work, presented by Chuck Lever, is a direct result of the Linux scalability project at the University of Michigan. This project aims to enumerate the problems facing network servers and ensure that Linux offers support greater than or equal to current production-level operating systems for network servers.

Because malloc() is a major concern for network servers, it is important that it be able to offer acceptable performance in situations that require multithreaded asynchronous I/O, low latency and high data throughput with a predictable response time, and a potentially unbounded input set of unpredictable requests. It has been found that the manner in which malloc() performs can have a significant effect in overall system performance. One example showed a 6x performance degradation when the iPlanet LDAP server ran on four-way hardware with a native implementation of malloc().

To study the performance of malloc(), three benchmarks were created. The first compared the elapsed runtime of N threads accessing the same heap concurrently. It was found that Linux outperformed Solaris for N greater than two. The second benchmark, which tested unbounded memory consumption, allocated an object in one thread and free the same object in the second thread. It was found that Linux has no pathological heap growth in this situation. The final benchmark tested data placement within the heap by allocating a data object normally with malloc() and then letting several threads write into it many times. Bad data placement causes poor performance, especially on SMP hardware. Linux's version of malloc() aligns data to eight-byte boundaries, which resulted in widely varying application performance on CITI's four-way test machine. In conclusion, the version of malloc() offered with glibc 2.1 showed overall acceptable performance on two- and four-way hardware, but still needs work and special attention to reduce performance and scalability problems caused by sloppy data placement.

CLOSING SESSION

New Horizons for Music on the Internet

Thomas Dolby Robertson

Summarized by Josh Simon

Thomas Dolby is a musician (you'll probably remember him from "She Blinded Me with Science!") who's been working for at least 20 years on integrating computers into music. (One historical tidbit: The drums in "Science!" were actually generated by a discotheque's light-control board.)

Dolby is one of the founders of Beatnik (<*http://www.beatnik.com/*>), a tool suite or platform to transfer descriptions of the music, not the music itself, over the Internet. For example, the description would define which voice and attributes to use, and the local client side would be able to translate that into music or effects. This effectively allows a Web page to be scored for sound as well as for sight.

For example, several companies have theme music for their logos that you may have heard on TV or radio ads. These companies can now, when you visit their Web sites, play the jingle theme without needing to download hundreds of kilobytes, merely tens of bytes. Similarly, a Web designer can now add sound effects to her site, such that scrolling over a button not only lights the button but plays a sound effect. Another use for the technology is to mix your own music with your favorite artists, turning on and off tracks (such as drums, guitars, and vocals) as you see fit, allowing for personalized albums at a fraction of the disk space. (In the example provided during the talk, a 20K text file would replace a 5MB MP3 file.) In addition to the "way cool" and "marketing" approaches, there's an additional educational component to Beatnik. For example, you can set up musical regions on a page and allow the user to experiment with mixing different instruments to generate different types of sounds.

The technical information: Beatnik combines the best of the MIDI format's efficiency and the WAV format's fidelity. Using "a proprietary key thingy" for encryption, Beatnik is interactive and cross-platform, providing an easy way to author music. And because the client is free, anyone can play the results. The audio engine is a 64-voice general MIDI synthesizer and mixer, with downloadable samples, audio file streaming, and a 64:2 channel digital mixer. It uses less than 0.5% of a CPU per voice, and there are 75 callable Java methods at runtime. It supports all the common formats (midi, mp3, wav, aiff, au, and snd), as well as a proprietary rich music format (rmf), which is both compressed and encrypted with the copyright. RMF files can be created with the Beatnik Editor. (Version 2 is free while in beta but may be for-pay software in production.) The editor allows for access to a sound bank, sequencer, envelope settings, filters, oscillations, reverbs, batch conversions (for example, entire libraries), converting loops and samples to MP3, and encryption of your sound. And there is an archive of licensable music so you can pay the royalties and get the license burned into your sample.

Web authoring is easy with the EZ Sonifier tool, which generates JavaScript; middling with tools like NetObjects' Fusion, Adobe GoLive, and Macromedia Dreamweaver; and hard if you write it yourself, though there is a JavaScript-authoring API available for the music object.

Beatnik is partnered with Skywalker Sound, the sound-effects division of Lucasfilms Ltd.

BOF SESSION

Workplace Issues for Lesbian, Gay, Bisexual, Transgendered and Friends

Summarized by Chris Josephes and Tom Limoncelli

Although the first submitted name for this BoF suggested that it was for "sysadmins," it was well-attended by managers, engineers, programmers, and anyone else who felt like attending. The LGBT BoF has had a long history at USENIX and LISA conferences. The crowd was an even mix of newcomers and conference veterans. The purpose of the session was to give people to opportunity to talk about their work environments, and to provide the opportunity to network with other attendees whom they may not have otherwise met during the conference.

Everyone introduced themselves and the companies they work for. Then they talked about how their employers have handled issues facing LGBT employees and related experiences they may have had when dealing with their employers. As it is becoming harder to find qualified people, more companies are adapting their benefits to the LGBT community in hopes of attracting new talent and retaining existing employees.

Of the 34 attendees, most reported that their employers established a nondiscrimination policy that included sexual orientation. On top of that, some employers also offered domestic-partnership benefits for registered partners. Another issue employers are trying to address is maintaining a safe, friendly, nonthreatening environment for employees by implementing peering programs, diversity training, and employee groups.

Andrew Hume, USENIX vice president, attended the BoF to welcome everyone and assure us that USENIX is committed to creating a space that is inviting and supportive to all attendees. His comments were well-received.

After the introductions were finished, the BoF became an open floor. We talked more in depth about the issues of employment and some of the recruiting plans some firms were offering. Several attendees, all from one large company, reported that they now have a recruitment effort targeting the LGBT community since they feel their accepting work environment is one of their competitive advantages. Someone pointed out which BoF attendees were presenting papers at the conference, so that others could attend and lend support. Ideas for improving attendance at future BoFs were brought up, including a couple of suggestions for making sure the male/female ratio was more representative of the conference attendance.

When it was time for the BoF to officially end, everyone agreed to informally get together at one of the hospitality suites scheduled for that night.

Trip Report

USENIX ANNUAL TECHNICAL CONFERENCE

by Peter H. Salus

<peter@matrix.net>

I had a great time at the 25th Anniversary USENIX conference in San Diego. Oh, boy! San Diego! Right. I didn't even get off the hotel premises for nearly three days.

After all, there were Dennis Ritchie, Ken Thompson, Bill Joy, Rob Pike, Kirk McKusick, Eric Allman, Tom Christiansen, Elizabeth Zwicky, Margo Seltzer, Sam Leffler, Jim Gettys, Clem Cole, Evi Nemeth, Mike Ubell, Teus Hagen, Rich Miller, Oz Yigit, Miguel de Icaza, and over 2000 others inside the hotel.

In fact, I only got to about two papers/presentations a day. Take Thursday, the "middle" day of the conference. From 9:00 to 11:00 am, I listened to Ed Felten of Princeton University talk about the Microsoft trial. As he was under DoJ secrecy rules (he was adviser to and expert witness for the DoJ), there was much he couldn't say. What he could say was fascinating. Then I walked around the exhibits, and chatted. Then I went to a publications-committee meeting where I was a guest of the USENIX board. Then I was on the Dr. Dobb's Webcast for over an hour, to be succeeded by Linus Torvalds. So I chatted with Linus, his wife, and their two little girls. Then I sat down with some folks from Sleepycat to learn about embedded databases. And it's now after 5:00 pm. At 6:00, I went to the celebratory reception; at 8:00, I went to hear Linus at the Linux BoF. At nine, I went to the "Old Farts' BoF." At 10:30 pm I met my wife, who wanted to go out for dinner. I was too tired. We stayed in the Marriott.

On Wednesday morning I had the pleasure of witnessing the awarding of the "Flame" (for lifetime achievement) to the late Rich Stevens, with Rich's wife, children, and sister getting a five-minute standing ovation from the attendees. Then Bill Joy shared his thoughts about the future of computing.

Bill traced his beginnings in the field – just about 25 years ago at Berkeley – and traced the origins of "open source" to the bases of university research and of UNIX on the PDP-11. While there was the question of whether doing software is "research," the common answer was "yes," and as researchers publish results, there could be no property rights adhering to that research. Lately, largely because of commercial and industrial contributions, there are more and more "entanglements."

Bill noted that he had written "a really boring Java book." My guess is that he was referring to *The Java Language Specification* by Joy, Steele, Gosling, and Bracha. As I really liked the first edition (1996) and found the second edition (2000) even better – when was the last time the second edition of a reference book was smaller than the first? – I think I'd disagree.

One of Joy's more interesting comments turned on the fact that UNIX was inherently reliable because of its modularization. The consequence is that "Microsoft is clearly foolish" in attempting to make all of its applications integral and the construct monolithic. "Microsoft is beyond retrograde," Bill said. "All interfaces should be published."

Turning to the future, Bill scorned the notion of the death of Moore's law. In fact, he thinks that we might see another "ten-to-the-sixth" improvement over the next 30 years, just as machinery has improved a million times over the past 30. Bill thinks that molecular computing will enable us to come to grips with the "grand challenge problems" like those of cell biology. Some of these steps will come about through algorithmic improvements, some through greater emphasis on remote storage and fast transmission.

However, he pointed out, the more that's done remotely, the higher the toll for the round-trip, even with high-speed optical connections. "The Internet isn't about packets," he said. "It's about end-to-end."

On the client-server side, Bill said that he saw six Webs in the future:

the "near web," which we currently use;
the "far web," the entertainment for couch potatoes;
the "here web," of the pocket device and the cell phone;
the "weird web," involving smart clothing and voice-activated cars;
and two "invisible" webs: "e-business" and "truly pervasive."

There was lots more. It was a fine hour.

Then I spent over an hour on the Dr. Dobb's Webcast.

Wednesday night there was a four-hour BSD BoF (that's right) organized by Kirk McKusick. An hour each of OpenBSD, FreeBSD, NetBSD, and BSDI. I sat through over an hour of it. While there were pockets of enthusiasts cheering and jeering, it was a generally highly intelligent, well-informed group of about 400. And I thought it exciting to see them all together. A doff of my cap to you, Kirk.

Thursday morning, as I mentioned, I went to hear Ed Felten. While most folks know many of the details of the case, I found listening to the recollections of a participant truly fascinating. My personal feeling is that the case is more about economics and business than about technology, but there are (clearly) two technical queries of significance: (1) is there a technical advantage to tying Windows to the browser? and (2) can they be disentangled without injury to either?

The clear answer to (1) is "no" (there is, of course, a business advantage for Microsoft) and Felten himself demonstrated the answer to (2) in court. In fact, advocates of the small kernel (like Linus Torvalds) as well as "old-timers" (like Bill Joy) all shun the interwoven monolithic monstrosities produced by Microsoft.

Went off for another hour of Webcast.

The "Old Farts' BoF" on Thursday night was very well attended. Ken Thompson, Lou Katz, Greg Rose, Dennis Moomaw, Clem Cole, and I were among the recollectors. Toward 10:00 pm, Professor Arun K. Sharma, head of computer science and engineering at the University of New South Wales, announced a "drive" to raise $A2M (= $US1.2M) to establish a John Lions Chair of Operating Systems at UNSW. As I considered John a teacher and a friend, I handed Arun my check for $1,000 on the spot. I found it very moving.

Friday morning I was spellbound as Rob Pike, who is always exciting to hear, spoke about quantum computing. This is the kind of paper that differentiates a real conference from a hawker's sideshow: Heisenberg, Schroedinger, integral signs, real equations. Rob said that we should rid ourselves of the notion that "the elements of information are independent" and get used to concepts like "conservation of information." "A quantum computer is probabilistic," he remarked. "It's not going to happen soon," but it will happen.

Now we can turn it over to Gibson, Sterling, Stephenson, and Vinge . . .

A fine conference. I can't wait till the 30th Anniversary.

The 25th Anniversary Reception was a good party!