The following paper was originally published in the
Proceedings of the Eleventh Systems Administration Conference (LISA '97)
San Diego, California, October 1997

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Shuse At Two: Multi-Host Account Administration

Henry Spencer – SP Systems

ABSTRACT

The Shuse multi-host account administration system [1] is now two years old, and clearly a success. It is managing a user population of 20,000+ at Sheridan College, and a smaller but more demanding population at a local ''wholesaler'' ISP, Cancom. This paper reviews some of the experiences along the way, and attempts to draw some lessons from them.

Shuse: Outline and Status

Shuse [1] is a multi-host account administration system designed for large user communities (tens of thousands) on possibly-heterogeneous networks. It adds, deletes, moves, renames, and otherwise administers user accounts on multiple servers, with functionality generally similar to that of Project Athena's Service Management System [2].

Shuse uses a fairly centralized architecture. All user/sysadmin requests go to a central daemon (*shused*) on a central server host, via *gatekeeper* processes on that host, which handle network interaction and authentication, and insulate the daemon from user misbehavior. *Shused* itself handles all database updates, regenerates derived files like the *passwd* file as necessary, and makes calls to various helper processes. *Shused* is single-threaded using an event-loop organization; long-running operations (e.g., updates to ''slave'' servers) are done by an auxiliary daemon (*shuselace*, which talks to *shused* using roughly the same command interface employed by users), so that *shused* itself is not tied up and unresponsive during such operations. *Shuselace* performs operations on a slave server by invoking (via *telnet* and *inetd*) a *shusetie* process there.

Shused keeps the entire user database in memory for fast response (RAM is cheaper than database packages). Crashproofing, initial startup, and emergency manual changes require a copy of the database on disk, but the presence of the memory copy allows optimizing the disk copy for quick updates rather than rapid bulk access. The disk copy is stored as one file per user, using a simple text format with each line containing a field name and a field value.

Shuse was written essentially entirely in Expect [3, 4] (an extended version of Tcl [5, 6]), a decision we have not regretted. A few tiny auxiliary programs written in C do jobs that are not feasible in Expect, and there are some shell files around the periphery as well. Performance has been quite satisfactory after a few early problems were resolved. Maintenance and enhancement have been greatly eased, and portability has been trivial.

After some trying moments early on, Shuse is very clearly a success. In mid-autumn 1996, Sheridan College's queue of outstanding help-desk requests was two orders of magnitude shorter than it had been in previous years, despite reductions in support manpower. Favorable comments were heard from faculty who had never previously had anything good to say about computing support. However, there was naturally still a wishlist of desirable improvements.

At around the same time, ex-Sheridan people were involved in getting Canadian Satellite Communications Inc. (''Cancom'') into the ISP business as a ''wholesaler'' ISP, supplying connectivity and resources to a network of retail dealers in mostly-remote areas. They decided they wanted to use Shuse.

Evolution

Cancom's use of Shuse required tracking slightly different information and generating slightly different outputs. This exposed a fair number of Sheridan-specific assumptions, some of which were easier to cure than others.

Merely adding new fields to the database was fairly trivial, thanks to the early decision to adopt a highly extensible format.

Changes to the Shuse code were also required, however, and since the Sheridan version was inevitably evolving in parallel, periodic code merges have been required ever since. This has been a bit tedious but not fundamentally difficult; we're still discovering which things need to be parameterized so that configuration files can customize them to customer needs. We don't think we could reasonably have anticipated most of them; indeed, some of the early attempts at such anticipation have gone unused.

Assorted additional facilities also had to be added, such as a Shuse implementation of the *poppassd* interface that lets non-shell users change their passwords. (This was originally done for Cancom, but turned out to be of considerable interest to Sheridan as well – a pattern that has held for a number of the changes.) We initially tried to make the usual freeware *poppassd* implementation talk to Shuse, but after a

number of problems (including one security breach via a core dump!), we gave up in disgust and wrote our own in Expect. It turned out to be shorter and much more robust and versatile than the original C code (which is basically trying to do an Expect-like job without the proper tools).

A more substantial user interface that also had to be added was a menu-oriented interface for Cancom's dealers, so that routine account administration could be delegated (subject to appropriate restrictions) to them. Original ideas of GUIs based on Tk [6] had to be shelved in favor of a very simple text-based menu system because of a tight schedule, unpredictable variations in user equipment, and the limitations of "long thin" communications links.

The dealer interface is not glossy and elegant, but it has worked out well. We were worried about response time, especially given those communications links, but the dealers have been so happy at being able to do their administration themselves – getting results in seconds instead of hours or days – that a bit of slowness hasn't yet elicited any critical comment. The one real blemish of the current design is that the dealer program knows too many things that *shused* also knows, so any change to such things has to be made in several places. Fixing that will require more complex provisions for user interfaces to obtain such information from *shused*, perhaps as downloaded Tcl code – a promising approach, but also a complicated one.

The text-based dealer interface naturally speaks English. This being Canada, soon after the dealer interface went into operation we got asked "what about a French version?" Intense distaste for the thought of maintaining two separate versions of the same code, plus a slight possibility of needing more languages in future (Cree has been mentioned...) led us to invent a more general solution: a message-catalog system for Tcl [7]. This hasn't seen enough use yet for a good evaluation, but it seems adequate to do the job.

We note that although it was originally envisioned that there would be multiple user-interface programs talking directly to *shused*, in practice all interfaces to date have been built on top of the *shusedo* program, which simply sends a single command to the daemon and outputs the response. A simple command-line interface like this lends itself well to the construction of more complex and more interactive interfaces; the reverse is *not* true.

Decentralization

The original Sheridan version of Shuse relied very heavily on NFS – in particular, it shared the */usr/local/shuse* tree that way – and distributed the *passwd* file (etc.) using NIS/YP. Cancom did not particularly want to do either, mostly for reasons of security. Sheridan wanted to continue using NIS but was having second thoughts about having an important administrative area widely NFS-mounted. Considerable effort was needed to adapt Shuse to a more loosely-coupled environment.

We had to make a concerted search for places where Shuse components explicitly or implicitly relied on shared filesystems or NIS, and fix them all. Inevitably, one or two were discovered only after the code was put into service. (One particular complication was that Sheridan was still running with a shared */usr/local/shuse* temporarily. It was not enough that the new code work correctly with a non-shared */usr/local/shuse*: it had to work whether */usr/local/shuse* was shared or not, and NFS's odd treatment of *root* caused minor difficulties here.)

The *telnet* connection which *shuselace* uses to give orders to *shusetie* is unsuited to bulk data transmission, which originally was done via NFS file sharing. Bulk data transfers are now done by FTP, using Expect to drive the Unix *ftp* utility.[1] We picked FTP because it was already installed and the networks involved are reasonably well controlled; a future shift to cryptographic authentication, e.g., via *ssh/scp*, is being considered for Cancom in particular.

Finding and fixing the more subtle dependencies on shared files and NIS was tedious, but it has had useful side effects. For example, fixing Shuse's password changers to consult *shused* for the old encrypted password, rather than doing their own accesses to the *passwd* file or invoking *ypmatch*, has also made them compatible with shadow password files.

The original Shuse updated the *passwd* file by simply generating a copy of the new file in a known location, whence a *cron* job regularly picked it up and shoved it into NIS. Without NIS, Shuse had to do its own updates of the *passwd* file on slave servers, which is harder than it sounds because of the shortage of decent programming interfaces for this.[2] We ended up telling *vipw* to use *ed* as its text editor, and driving *ed* via Expect. This had a few problems of its own (such as discovering that we could easily overdrive a BSD/OS pseudo-tty to the point where it would lose characters), but with careful checking and some judicious use of some of Expect's more obscure features, in the end it worked. (The pseudo-tty overdriving problem, in particular, was solved by insertion of a 1 ms delay every 40 characters, which is easy with Expect.) The resulting code is so paranoid that it has

[1] The *ftp* program is another one of those wonderful utilities which *knows*, by God, that it is running interactively, and completely neglects to provide any sort of reasonable *programming* interface. Fortunately, Expect deals with this reasonably well, at the cost of some tedious experimenting to uncover all the likely interactive messages.

[2] As with FTP and quotas, the *passwd*-file user interfaces are a disgrace to Unix, since they typically insist on running interactively and cannot be programmed without resorting to Expect.

caught various unanticipated problems (e.g., *vipw* running out of disk space).

The lack of shared filesystems necessarily means replicating the Shuse software on the various hosts. This has been somewhat error-prone, and we've been working on reducing its problems. We've been progressively eliminating shared control files that have to be kept consistent: about the only ones remaining are authentication keys and Shuse's configuration files. Some of the eliminated files never really had to be shared; others are necessary but are small enough to be transmitted each time. We've also been making an effort to reduce the number of code files that have to be present on a slave server, although there are limits to this.

More fundamentally, Shuse is now taking on the responsibility of updating the remote copies of itself! This is arguably a re-invention of the wheel, since existing software packages like *rdist* [8] can handle this sort of thing. However, one of the customers wanted it and was willing to pay for it, so we did it. It was quite straightforward, using FTP for data transfers, except for the need to make careful provision for backing out of an update that happens to break the slave-server side of the software.

Since the remote-update facility now exists, it's also being used as a substitute for other inter-machine propagation mechanisms. For example, Cancom is using it to propagate updates to */etc/group*.

Could all this effort have been avoided with greater forethought about loosely-coupled systems? Probably most of it, yes, but there were good reasons why it wasn't done that way the first time. For one thing, priorities have changed with experience: some of the original approaches had apparent advantages that have not proved significant in practice.[3] For another, the new mechanisms have added substantial complexity in places, and time constraints weighed heavily in the early development of Shuse [1].

Internal Cleanup

Every time we've put effort into cleaning up and generalizing Shuse's innards, we've regretted not doing it sooner. Many things have become easier this way; many of the remaining internal nuisances are concentrated in areas which haven't had such an overhaul lately.

The most fundamental area of internal change has been the protocols used to request updates on the slave servers. The original Shuse design envisioned a very simple and appealing approach: *shused* would distribute a description of the way things were supposed to be, and the slave servers would compare this to the actual situation and do any changes needed to make reality match the description. This has the advantage of being extremely robust in general, and completely crash-proof in particular: no matter who crashes and when, if everybody is eventually up for long enough, the slave servers will synchronize with *shused*'s current opinion on how things should look.

Unfortunately, too many details of real account maintenance did not fit this description-based model very well. There were a few early signs of difficulty, but the real killer was the need for coordinated activity by two slave servers when moving a user from one server to another.

A number of difficulties were cleared up by breaking down and conceding that some operations would simply have to be done in tight lockstep, with *shused* retaining a to-be-done list and checking items off as they were completed. Making such operations crash-proof is not fundamentally difficult, given the facility for planting ''at commands'' (to be executed at specific times) in a user's database entry, although the details unfortunately ended up being rather complex.[4] This was first done for user moves, and has since been extended to renames and quota settings. User deletion, implemented originally using the description-based model, will probably move to a lockstep implementation.

On the other hand, serious consideration is being given to moving back towards the description-based model for some things. In particular, disk-quota updates are currently done as lockstep operations, but this falls down badly in situations like restoring a user filesystem from a backup. Quotas probably *should* be handled with the description-based model, and we're going to look harder at this.

The moral we draw from this is that one should not overlook the need for feedback: the key oversight in the original description-based model was that even though *shused* is always leading and the slave servers are always following, some operations do require *shused* to know exactly when a change takes place on a slave server.

Much of the other internal-cleanup work has been focused on what might be called improving the software engineering of Shuse's innards. Useful facilities have been generalized and encapsulated to make them easier to use for multiple purposes. Shared knowledge has been eliminated, e.g., by making data-transmission formats self-describing. Bright ideas that turned out to be mistakes have been ripped out and replaced by simpler approaches. None of this should

[3] For example, we'd originally hoped to avoid having to use shared secrets for authentication, by relying on permissions of shared directories instead. Shared secrets proved necessary for other reasons.

[4] Getting a single lockstep operation done in a fully crash-proof way involves four or more at-commands interwoven in a pattern resembling a database two-phase-commit protocol. The internals documentation calls this process ''The Dance of the At-Commands.''

really require comment, except that it's so seldom actually *done* on real software. We've found that effort spent on this typically pays off handsomely, by making later changes easier.

Administration

Although it would be nice if automated-sysadmin software didn't itself require administration, it does. The interfaces used for this are sometimes skimped on. Shuse has needed improvements in several areas of its administrative interfaces.

One area that fortunately *hasn't* needed much improvement is operational robustness. There is an obvious vulnerability in a single central server process running on a single central server host: what happens if the process or the host crashes? We originally decided that it was better to spend effort on eliminating process and host unreliability than to devise elaborate distributed protocols to cope with it. This decision has been amply vindicated.

Making the host reliable hasn't required much effort. Making the *shused* process reliable did take some. In particular, early development versions of *shused* died whenever their innards signalled an error, on the theory that it might have caused corruption of the database. This got fixed before entry into production: while there is *potential* for database corruption, most real error signals denote nothing more fundamental than a minor bug in the particular request being executed, and logging the problem and carrying on is vastly preferable to falling over dead.

Given this, *shused* has proved reliable enough that we've done nothing at all about automated recovery from catastrophic failures: if it happens, the staff notice and restart the daemon. While new versions of *shused* have occasionally fallen over suddenly, once such last-minute problems are resolved, reliability has been high. The *shused* process running at the time this is being written, in early September, has been running since a development-induced restart at the end of July.

A related but more subtle problem is that early versions of *shuselace* occasionally died under mysterious circumstances. This was much more subtle than having *shused* die, because *shused* was still handling interactive commands and database updates properly, only the updates didn't propagate out to the slave servers. Although the problem hasn't happened recently, *shused* now guards against this and other *shuselace* failures by ''pinging'' *shuselace* regularly and complaining if there is no response. Again, the problem hasn't been frequent enough to justify automated handling (which would be slightly awkward due to implementation details); the fix is to shut down and restart *shused*.

An area that was, for a long time, rather less satisfactory was trouble reporting. The original *shused* design lacked any orderly way of reporting difficulties to the sysadmins. Theoretically one could regularly inspect the log file, but in practice this didn't get done very much.

The problem was greatly aggravated with the arrival of lockstep operations like renaming a user, where the need for slave-server cooperation means that actual execution of the operation may be arbitrarily delayed pending availability of the necessary slave servers. The command executed by the user merely queues up the operation, and originally there was no systematic way of reporting success or failure of the ultimate execution. This was particularly troublesome because such operations *usually* succeed quickly, and so one gets out of the habit of checking on them.

After a period of just muddling along, this got fixed in the obvious way: the results of delayed operations, and independently-discovered indications of trouble, are reported by mail messages. This wasn't quite as simple as it looked, because an administrator who moves 500 users (not at all unusual in a user population of 20,000+) does not want to get 500 separate mail messages reporting success. An ambitious design for merging similar messages was thought out but shelved (although there are implementation hooks for it) in favor of a simple timeout mechanism, which just holds onto non-urgent reports briefly in hopes of being able to send more than one report in a single message. The details are still being tuned, but this seems generally adequate. We should really have thought about this, and done it, at the time the first delayed operations appeared (if not earlier).

While the *shused* log file (which incorporates log entries sent in by other components of the software, like *shusetie*) is useful for debugging, it's large, and less than ideal when it comes to answering questions like ''who last changed user *xyz*'s disk quota?'' A little bit of historical information is kept in user database entries as it stands – for example, there is a timestamp field identifying the time and instigator of the last password change – but this isn't always adequate. Work is now underway on a general facility for holding a configuration-specified amount of command history in each user database entry, so that information on at least the recent changes will be *conveniently* available when needed.

Conclusion

Shuse continues to evolve, partly because we're still learning what it needs to do. Expanding operations to a second user community has made a lot of learning happen fairly suddenly. We're also belatedly recognizing things we should have noticed a while back, such as the need for better administrative interfaces.

Acknowledgements

Sheridan College in general, and Cheri Weaver in particular, got Shuse started. John Barber of Sheridan and Doug Berry of Cancom have supported its

continuing evolution. A number of people, most notably Trevor Stott and Doug Berry, have used Shuse at length and have been patient as problems were found and fixed.

Availability

Bad news: it's still not freely available, alas. Sheridan continues to be interested in the possibility of commercial marketing, although nothing organized has happened yet. If you're interested in getting Shuse, contact the author, and he'll pass your inquiry on to the right people.

Author Information

Henry Spencer is a freelance software engineer and author. His degrees are from University of Saskatchewan and University of Toronto. He is the author of several freely-redistributable software packages, notably the original public-domain *getopt*, the redistributable regular-expression library, and the *awf* text formatter, and is co-author of C News. He is currently immersed in the complexities of implementing POSIX regular expressions. He can be reached as henry@zoo.toronto.edu .

References

[1] Henry Spencer, ''Shuse: Multi-Host Account Administration,'' *Proceedings of the Tenth Systems Administration Conference (LISA X)*, September 1996 (Chicago), Usenix Association 1996.

[2] Mark A. Rosenstein, Daniel E. Geer, & Peter J. Levine, ''The Athena Service Management System,'' *Proceedings of the Usenix Technical Conference,* Winter 1988 (Dallas), Usenix Association 1988.

[3] Don Libes, ''Expect: Curing Those Uncontrollable Fits of Interaction,'' *Proceedings of the Usenix Technical Conference,* Summer 1990 (Anaheim), Usenix Association 1990.

[4] Don Libes, *Exploring Expect*, O'Reilly & Associates 1995.

[5] John K. Ousterhout, ''Tcl: An Embeddable Command Language,'' *Proceedings of the Usenix Technical Conference,* Winter 1990 (Washington), Usenix Association 1990.

[6] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley 1994.

[7] Henry Spencer, ''Simple Multilingual Support for Tcl,'' *Proceedings of the Fifth Annual Tcl/Tk Workshop,* July 1997 (Boston), Usenix Association 1997.

[8] Michael Cooper, ''Overhauling Rdist for the '90s,'' *Proceedings of the Usenix Technical Conference,* Winter 1987 (Washington), Usenix Association, 1987.