HLFSD: Delivering Email to Your $HOME

Erez Zadok - Columbia University
Alexander Dupuy - System Management ARTS

HLFSD: Delivering Email to Your $HOME

Erez Zadok – Columbia University
Alexander Dupuy – System Management ARTS

ABSTRACT

We consider the problem of enabling users to access their mailbox files from any host on our local network, and not only on one designated “home machine”. We require a solution which will not introduce any new single points of failure, force us to modify mail transfer agents and user agents, or require changes to the operating system kernels. In other words, minimize the amount of work needed by system-administrators and users. Our solution is to deliver mail into the users’ home directories, which are exported via NFS[20,25] to all of the machines on our network. We wrote a small user-level NFS server implementing a single symbolic link that references the home directory of a user, either the one who accessed it, or by name, with a fallback reference in case of failures. This enables electronic mail to be delivered directly into the user’s home directory, which is already accessible from any machine on the network. Although we have used our server primarily for mail delivery redirection, it can be used to redirect spooled faxes, access to `/tmp`, etc.

Introduction

Electronic mail has become one of the major applications for many computer networks, and use of this service is expected to increase over time, as networks proliferate and become faster. Providing a convenient environment for users to read, compose, and send electronic mail has become a requirement for systems administrators (SAs).

Widely used methods for handling mail usually require users to be logged into a designated “home” machine, where their mailbox files reside. Only on that one machine can they read newly arrived mail. Since users have to be logged into that system to read their mail, they often find it convenient to run all of their other processes on that system as well, including memory and CPU-intensive jobs. For example, in our department, we have allocated and configured several multi-processor servers to handle such demanding CPU/memory applications, but these were under-utilized, in large part due to the inconvenience of not being able to read mail on those machines. (No home directories were located on these designated CPU-servers, since we did not want NFS service for users’ home directories to have to compete with intensive jobs. At the same time, we discouraged users from running demanding applications on their home machines.)

Many different solutions have been proposed to allow users to read their mail on any host. However, all of these solutions fail in one or more of several ways:

- They introduce new single points of failure
- They require using different mail transfer agents (MTAs)[15] or user agents (UAs)
- They do not solve the problem for all cases, i.e., the solution is only partially successful for a particular environment.

We have designed a simple filesystem, called the *Home-Link File System*, to provide the ability to deliver mail to users’ home directories, without modification to mail-related applications. We have endeavored to make it as stable as possible. Of great importance to us was to make sure the HLFS daemon, `hlfsd`, would not hang under any circumstances, and would take the next-best action when faced with problems. Compared to alternative methods, `hlfsd` is a stable, more general solution, and easier to install/use. In fact, in some ways, we have even managed to improve the reliability and security of mail service.

Our server implements a small filesystem containing a symbolic link to a subdirectory of the invoking user’s home directory, and named symbolic links to users’ mailbox files. An example, using pathnames from our environment, is depicted in Figure 1.[1]

The `hlfsd` server finds out the *uid*[2] of the process that is accessing its mount point, and resolves the pathname component `home` as a symbolic link to a subdirectory within the home directory given by the *uid*’s entry in the password file (see Table 1). If the *gid* of the process that attempts to access a mailbox file is a special one (called HLFS_GID), then the server maps the name of the <u>next</u> pathname

[1] In Figure 1, *~NAME* is the home directory of the user whose user-name is *NAME*; *~USER* is the home directory of the user with user-id *uid*.

[2] NFS uses effective *uids*.

component directly to the user's mailbox (Table 2). This is necessary so that access to a mailbox file by users other than the owner can succeed. The server has safety features in case of failures such as hung filesystems or home directory filesystems that are inaccessible or full.

On most of our machines, mail gets delivered to the directory `/var/spool/mail`.[3] Many programs, including UAs, depend on that path. `Hlfsd` creates a directory `/mail`, and mounts itself on top of that directory. `Hlfsd` implements the path name component called `home`, pointing to a subdirectory of the user's home directory. We made a symbolic link from `/var/spool/mail` to `/mail/home`, so that accessing `/var/spool/mail` actually causes access to a subdirectory within a user's home directory.

The rest of this paper is organized as follows. The second section discusses in detail the problems and limitations of other home-mail-delivery methods. The third section details the design of the *Home-Link File System* and the fourth section describes the implementation of `hlfsd`. The next section

[3]Other directories used for this purpose are `/var/mail` on SVR4, `/usr/mail` on other System V-based operating-systems, and `/usr/spool/mail` on BSD-based systems.

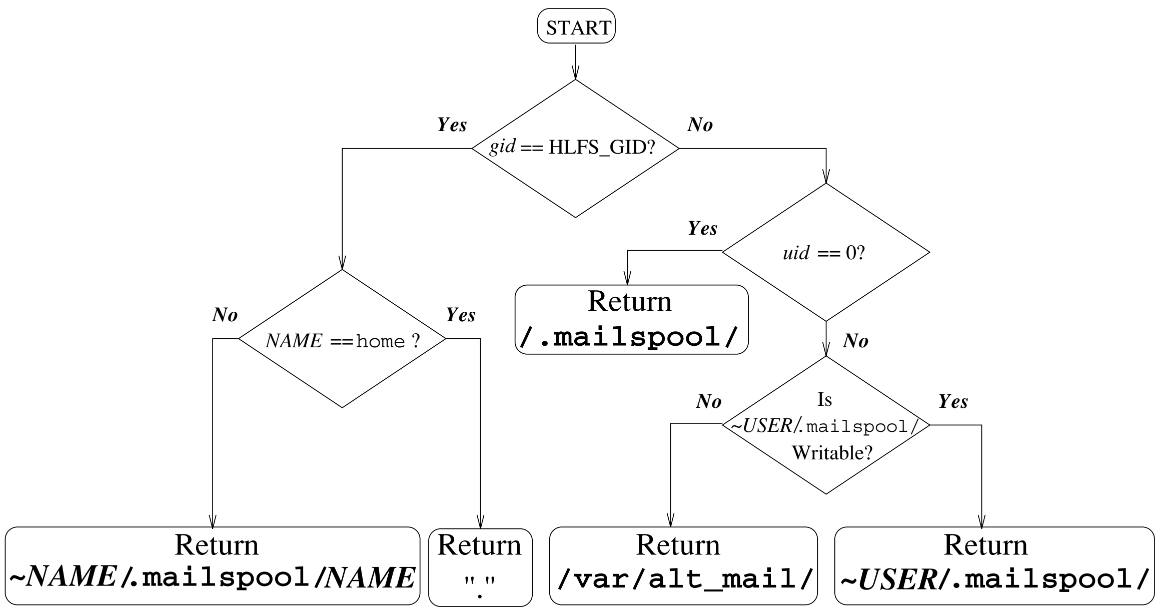

Figure 1: `Hlfsd` resolving the *NAME* component of `/mail/`*NAME*

Conditions: *uid* =`ezk`, *gid*„HLFS_GID, and `/users/ezk/.mailspool/` is writable.

Resolving component	Pathname left	Value if symbolic link
`/`	`var/mail/`*NAME*	
`var/`	`mail/`*NAME*	
`mail@`	`/mail/home/`*NAME*	`mail@   /mail/home`
`/`	`mail/home/`*NAME*	
`mail/`	`home/`*NAME*	
`home@`	*NAME*	`home@   /users/ezk/.mailspool`
`/`	`users/ezk/.mailspool/`*NAME*	
`users/`	`ezk/.mailspool/`*NAME*	
`ezk/`	`.mailspool/`*NAME*	
`.mailspool/`	*NAME*	
NAME		

Table 1: Resolving `/var/mail/`*NAME* to `/users/ezk/.mailspool/`*NAME*

evaluates our implementation. Related systems, conclusions, future directions, and alternative uses are described in the final two sections.

Background

This section provides an in-depth discussion of why available methods for delivering mail to home directories are not as good as `hlfsd`'s method.

Single Host Mail Spool Directory

The most common method for mail delivery is for mail to be appended to a mailbox file in a standard spool directory on the designated "mail home" machine of the user. The greatest advantage of this method is that it is the default method most vendors provide with their systems, thus very little (if any) configuration is required on the SA's part. All they need to set up are mail aliases directing mail to the host on which the user's mailbox file is assigned. (Otherwise, mail is delivered locally, and users find mailboxes on many machines.)

As users become more sophisticated, and aided by windowing systems, they find themselves logging in on multiple hosts at once, performing several tasks concurrently. They ask to be able to read their mail on any host on the network, not just the one designated as their "mail home."

Centralized Mail Spool Directory

A popular method for providing mail readability from any host is to have all mail delivered to a mail spool directory on a designated "mail-server" which is exported via NFS to all of the hosts on the network. Configuring such a system is relatively easy. On most systems, the bulk of the work is a one-time addition to one or two configuration files in `/etc`. The file-server's spool directory is then hard-mounted across every machine on the local network. In small environments with only a handful of hosts this can be an acceptable solution. In our department, with a couple of hundred active hosts and thousands of mail messages processed daily, this was deemed completely unacceptable, as it introduced several types of problems:

- **Scalability and Performance**: as more and more machines get added to the network, more mail traffic has to go over NFS to and from the mail-server. Users like to run mail-watchers[2,7] and read their mail often. The stress on the shared infrastructure increases with every user and host added; loads on the mail server would most certainly be high since all mail delivery goes through that one machine.[4] This leads to lower reliability and performance. To reduce the number of concurrent connections between clients and the server host, some SAs have resorted to automounting the mail-spool directory. But this solution only makes things worse: since users often run mail watchers, and many popular applications such as `trn`, `emacs`, `csh` or `ksh` check periodically for new mail, the automounted directory would be effectively permanently mounted. If it gets unmounted automatically by the automounter program[3], it is most likely to get mounted shortly afterwards, consuming more I/O resources by the constant cycle of `mount` and `umount` calls.
- **Reliability**: the mail-server host and its network connectivity must be very reliable. Worse, since the spool directory has to be hard-mounted,[5] many processes which access the spool directory (various shells, `login`, `emacs`, etc.) would be hung as long as connectivity to the mail-server is severed. To improve reliability, SAs may choose to backup the mail-server's spool partition several times a day. This may make things worse since reading or delivering mail while backups are in progress may cause backups to be inconsistent; more backups consume more backup-media resources, and increase the load on the mail-server host.

Distributed Mail Spool Service

Despite the existence of a few systems that support delivery to users' home directories,[6] mail delivery to home directories hasn't caught on. We believe the main reason is that there are too many programs that "know" where mailbox files reside. Besides the obvious (the delivery program `/bin/mail` and mail readers like `/usr/ucb/Mail`, `mush`, `mm`, etc.), other programs that know mailbox location are `login`, `from`, almost every shell, `xbiff`, `xmailbox`, and even some programs not directly related to mail, such as `emacs` and `trn`. Although some of these programs can be configured to look in different directories with the use of environment variables and other resources, many of them cannot. The overall porting work is significant.

Other methods that have yet to catch on require the use of a special mail-reading server, such as IMAP[16] or POP[17]. The main disadvantage of these systems is that UAs need to be modified to use these services – a long and involved task. That is

[4]Delivery via NFS-mounted filesystems may require usage of `rpc.lockd` and `rpc.statd` to provide distributed file-locking, both of which are widely regarded as unstable and unreliable. Furthermore, this will degrade performance, as local processes as well as remote `nfsd` processes are kept busy.

[5]No SA in their right minds would soft-mount read/write partitions – the chances for data loss are too great.

[6]AIX 1.2's `bellmail` for the IBM PS/2*s*[9], `/bin/mail` on SunOS for the Sun 386i machines, and `zmailer`[27].

why they are not popular at this time. See the section on mail-reading servers for more details.

Several other ideas have been proposed and even used in various environments. None of them is robust. They are mostly very specialized, inflexible, and do not extend to the general case. Some potentially lose or corrupt mail:

- **Automounters**: using an automounter such as `amd`[13] to provide a set of symbolic links from the normal spool directory to user home directories is not sufficient. UAs rename, unlink, and recreate the mailbox as a regular file, therefore it must be a real file, not a symbolic link. Furthermore, it must reside in a real directory which is writable by the UAs and MTAs. This method may also require populating `/var/spool/mail` with symbolic links and making sure they are updated. Making `amd` manage that directory directly fails, since many various lock files need to be managed as well (see the section on lock files). Also, `amd` does not provide all of the NFS operations which are required to write mail such as *write*, *create*, *remove*, and *unlink*.
- **$MAIL**: setting this variable to an automounted directory on the user's mail spool host only solves the problem for those programs which know and use $MAIL. Many programs don't, therefore this solution is partial and of limited flexibility. Also, it requires the SAs or the users to set it themselves – an added level of inconvenience and possible failures.
- **/bin/mail**: using a different mail delivery agent could be the solution. One such example is `hdmail`[6]. However, `hdmail` still requires modifying all UAs, the MTA's configuration, installing new daemons, and changing login scripts. This makes the system less upgradable or compatible with others, and adds one more complicated system for SAs to deal with. It is not a complete solution because it still requires each user have their $MAIL variable setup correctly, and that every program use this variable.

Why Deliver Into The Home Directory?

There are several major reasons why SAs might want to deliver mail directly into the users' home directories:

- **Location**: many mail readers move mail from the spool directory to the user's home directory. It speeds up this operation if the two are on the same filesystem. If for some reason the user's home directory is inaccessible, it isn't that useful to be able to read mail, since there is no place to move it to.[7] In some cases, trying to move mail to a non-existent or hung filesystem may result in mail loss.
- **Distribution**: having all mail spool directories spread among the many more filesystems minimizes the chances that complete environments will grind to a halt when a single server is down. It does increase the chance that there will be someone who is not able to read their mail when a machine is down, but that is usually preferred to having no one be able to read their mail because a centralized mail server is down. The problem of losing some mail due to the (presumably) higher chances that a user's machine is down is minimized in HLFS as described in the sections on alternate mail spool directories and avoiding hangs.
- **Security**: delivering mail to users' home directories has another advantage – enhanced security and privacy. Since a shared system mail spool directory has to be world-readable and searchable,[8] any user can see whether other users have mail, when they last received new mail, or when they last read their mail. Programs such as `finger` display this information, which some consider an infringement of privacy. While it is possible to disable this feature of `finger` so that remote users cannot see a mailbox file's status, this doesn't prevent local users from getting the information. Furthermore, there are more programs which make use of this information. In shared environments, disabling such programs has to be done on a system-wide basis, but with mail delivered to users' home directories, users less concerned with privacy who *do* want to let others know when they last received or read mail can easily do so using file protection bits. Lastly, on systems that do not export their NFS filesystem with `anon=0`, superusers are less likely to snoop around others' mail, as they become "nobodies" across NFS.

In summary, delivering mail to home directories provides users the functionality sought, and also avoids most of the problems discussed in the section that discusses the centralized mail spool directory.

Design

We named our file system the *Home-Link File System*, because that's all it does – provide symbolic

[7]This assumes that they can login to a different host. Some systems, such as HP-UX, do not allow login if they cannot `chdir` to the user's home directory.

[8]System V has the mail spool directory only group writable but that makes it more difficult to install other UAs or MTAs.

links to various files and directories in a user's home directory. The soft link has a fixed name, but unlike regular soft links, what it points to is a "dynamic" target depending on who accesses the symbolic link. The ideas that this filesystem represents are not limited just to handling electronic mail – that is only one particular application of this filesystem. See the sections on future work and alternative uses for other ideas.

Our key goals in designing HLFS were:

1. To provide every user with the ability to read mail from any host.
2. To ensure that all MTAs and UAs in use, as well as any other utilities which depend on the standard mail spool directory, face no problems from the change in the underlying filesystem.
3. To minimize the possibility of mail being lost or bouncing back to the sender.
4. To provide more privacy for users' mail files.

Since most sites provide access to users' home directories from any host, it made sense to store incoming mail there as well. That way, as long as users could log into a host and access their home directories via NFS, their mail would be accessible as well. This solved the first problem. Also, since users must login in order to read their mail, causing their home directories to be automounted, no extra directory mounts are required in order to begin reading mail.

The second problem was solved by making sure that the final access of the mail spool directory remains a "real" directory (not a read-only pseudo-filesystem provided by an automounter). All UAs access the spool directory for reading and writing the user's mail file and create lock files there.[9] That means that `/var/spool/mail` needs to be readable, writable, and searchable for UAs and MTAs so that lock files can be written and removed (also see the section on Lock files). For the purpose of writing the mail and lock files, a subdirectory inside the user's home directory is sufficient, since it is already owned by that user. Ensuring that users cannot access other users' files can easily be achieved by protecting this subdirectory.

In order not to change the behavior of programs like `comsat`[21,23], `from`, or `finger`, which are designed to read any user's mail, we implemented a special check for a designated group. If the effective *gid* of the process is the designated group, we assume that such a special program is executing, and `hlfsd` arranges to do the lookup not only of the real spool directory, but of the mailbox itself. See Table 2. Note that this method only supports read access without locking; however, this is sufficient for all programs that need to access other users' mailboxes. All that is required is to set these programs to be *setgid* to the designated group.

The third problem is solved by ensuring that all operations which might hang `hlfsd` are performed in the background, while still providing service in the parent or child process. Furthermore, if `hlfsd` is not running anymore, or if the user's home filesystem is full, mail gets delivered to an alternate

[9]Note that, in order to allow mail delivery to NFS-mounted mail spool directories, most vendors have modified the `/bin/mail` program to set its *uid* to that of the recipient when delivering mail. If a local delivery agent (LDA) on a system does not provide this behavior, the MTA must arrange to invoke it with the *uid* of the recipient – this can be done by a wrapper C program.

Conditions: *gid*=HLFS_GID for any *uid*.

Resolving component	Pathname left	Value if symbolic link
`/`	`var/mail/`*NAME*	
`var/`	`mail/`*NAME*	
`mail@`	`/mail/home/`*NAME*	`mail@` `/mail/home`
`/`	`mail/home/`*NAME*	
`mail/`	`home/`*NAME*	
`home@`	*NAME*	`home@` `.`
`./`	*NAME*	
NAME@		*NAME*@ ~*NAME*`/.mailspool/`*NAME*
~*NAME*/	`.mailspool/`*NAME*	
`.mailspool/`	*NAME*	
NAME		

Table 2: Specially resolving `/var/mail/`*NAME* to ~*NAME*`/.mailspool/`*NAME*

directory (see the section entitled "Alternate Mail Spool Directories"). A `cron` job (running several times a day in our department), looks at the alternate directory, and attempts to resend messages in it to their rightful owners. All that is incurred is a delay in mail delivery, which, in most cases, is no longer than the length of time between consecutive invocations of the lost-mail remailing script.

Having a special mail-spool subdirectory owned and controlled by the user also addresses the last problem, that of privacy. Users can change the protection bits on their mailbox directory inside their home directory so that it is readable and searchable only by the owner. Any other program or user, unless running as the superuser on the same host,[10] would not be able to find out whether a user has new mail, how much of it there is, or when it was last read.

Implementation of `hlfsd`

We used a prototype NFS server, and implemented only the operations that were needed. We generated NFS stubs using `rpcgen`. The server was developed first under SunOS version 4.1.2. This server was incorporated into the `amd` source tree, and we used some of `amd`'s sources as utility functions, since they are well-written to handle a variety of architectures and operating systems. (See the section on source code size, availability, and portability.)

The "Home-Link" File Service

This subsection includes technical details of the NFS operations and may be skipped. However, it provides an example of the design and implementation of a small special-purpose NFS server and may be of use to others.

HLFS is a read-only filesystem, and as such, all operations that require write access return the error code NFSERR_ROFS ("Read-Only Filesystem"): *setattr, write, create, remove, rename, link, unlink, symlink, mkdir,* and *rmdir*. Trivially implemented were the *null*, *root*, and *writecache* operations. We decided to have *statfs* return some value (all zeros in most cases). The *read* operation simply returns NFSERR_ACCES ("Permission Denied").

The remaining operations are the heart of this filesystem: *readdir*, *getattr*, *lookup*, and *readlink*.

Our server must distinguish between the directory and the link, so we assigned them different integers to serve as filehandles. Note that these need not be as complicated as the filehandles usually generated by NFS. They need only to be unique, and their value is meaningful only to the server.

[10]Or as the superuser elsewhere, if the filesystem is NFS-exported with `anon=0`.

The readdir Operation

Opening this directory returns the "." and ".." directories, and one symbolic link, `home`. Attempting to readdir on the symbolic link results in an NFSERR_NOTDIR. Anything else is a stale filehandle.

The getattr Operation

Getattr returns `r-xr-xr-x` for the "." and ".." directories. The link itself, named `home` by default, is protected as `rwxrwxrwx`. It does not matter for the link that it is world-writable. The modification and creation times for the link and directories are the startup time of the server. If the effective *gid* of the process is HLFS_GID, then some fixed valid attributes are returned. Any other filehandle given to `hlfsd` is considered stale and the NFSERR_STALE ("Stale Filehandle") result code is returned.

The lookup Operation

Obviously, we only allow looking up in the "." and ".." directories, both of which return the same values. Trying to lookup "in" the link results in an NFSERR_NOTDIR ("Not a Directory") error code. Any link not known to the server returns an NFSERR_NOENT ("No Such Entry") error, unless the *gid* of the requesting process is HLFS_GID and the name corresponds to a valid user. In this case the *username* for that user is used in the returned filehandle, allowing the readlink operation to return the correct link. Anything else is a stale filehandle.

The readlink Operation

This is the most important operation, the central point of this work. We get the *uid* number from the credentials sent with the RPC operation. We make sure that Unix Authentication or DES is used or else we return the NFSERR_PERM ("Not Owner") code.

If the *gid* of the accessing process is not HLFS_GID, the value we return for the symbolic link named `home`[11] is a string representing the home directory of the user whose *uid* we just found, concatenated with a fixed component name representing a subdirectory within it. We used a binary search on the lookup table to quickly get the right pathname. Different home directories for multiple password database entries with the same *uid* numbers may return any of the home directories. Only *uid* 0 is guaranteed to return "`/`" (see "Problems").

If the symbolic link is named `home` and the *gid* is HLFS_GID, we return a link to ".", which causes `hlfsd` to be used to resolve the next pathname component. This is designed to maintain functionality of programs such as `from`. If the symbolic link is not named `home` and the *gid* of the accessing process is HLFS_GID, we return a value pointing to the

[11]The name of the symbolic link is configurable.

user's mailbox file in their mail spool directory. To do this, we extract the *username* from the filehandle, which was returned by the lookup operation (see Table 2).

Trying to readlink on one of the two directories results in an NFSERR_ISDIR ("Is a directory") error. Anything else is a stale filehandle.

Execution Flow

At initialization time, `hlfsd` creates a UDP service, and forks a child. The child builds the *uid* lookup table, sets up signal handlers, and interval timers. The signal handlers are meant to reload the lookup table at expiration time of the interval timer, or when a SIGHUP is sent to the server (presumably by a superuser). A special cleanup handler is setup for SIGTERM, to ensure the server terminates cleanly. Then the `svc_run()` routine is invoked.

Meanwhile the parent waits for the child to initialize. When it does, the parent mounts the server on the mount point. Of utmost importance is to make sure the attribute cache is turned off. If the attribute cache is not turned off, successive accesses to `/mail/home` would return previously computed pathnames pointing to another user's mail, resulting in mail loss or misdelivery. If it is not possible to turn off the attribute cache, `hlfsd` will exit. However, the SA has the option to force `hlfsd` to continue running and set the attribute cache to as short an interval as possible (see "Problems"). At this point the parent terminates, leaving the child to run.

When an interval timer goes off (SIGALRM) or a SIGHUP is sent to the server, the server forks a child that continues serving, while the parent reloads the lookup table. When the parent is finished loading, it sends a SIGKILL to the child process, and resumes serving. When a SIGTERM is received, the server forks a child that continues serving, while it tries to unmount the filesystem. If and when that succeeds, both parent and child exit.

As mail service is very important, we wanted to make `hlfsd` as robust as possible. We could have designed it as another `amd` "filesystem type", but decided that a separate daemon provides better reliability and faster service. In general, we try to do as much as possible: we make sure filesystems are accessible and contain some disk space to have mail delivered there. Where directories are expected we make sure there are no files by these names; where symbolic links are expected, we make sure there are no real files or directories with the same name. Whenever possible, we create directories, with proper ownership and permissions. We even check that the mount point for `hlfsd` is world readable and executable, since if it isn't, `getwd("..")` might fail.

Alternate Mail Spool Directories

`Hlfsd` tries to ensure that the user's home directory is accessible. Periodically it also tests that it can be written into (see the section on disk space problems). If for any reason a failure occurs, `hlfsd` repoints the symbolic link for that user to an alternate local directory, which is presumably highly available. We use `/var/spool/alt_mail` in our environment. See Table 3.[12]

When `hlfsd` starts up, and before it mounts itself on top of the mount point, hiding anything that is underneath, `hlfsd` creates a fixed symbolic link to the alternate spool directory (if it does not exist already). This is done so that `/var/spool/mail` would not be a "dangling" symbolic link, and points to a real directory at all times, even after `hlfsd`

[12]In the conditions for Table 3, ~*USER* is the home directory of the user with user-id *uid*.

Conditions: Any *uid*, *gid*,, HLFS_GID, and ~*USER*`/.mailspool/` is <u>not</u> writable

Resolving component	**Pathname left**	**Value if symbolic link**
/	var/mail/*NAME*	
var/	mail/*NAME*	
mail@	/mail/home/*NAME*	mail@ /mail/home
/	mail/home/*NAME*	
mail/	home/*NAME*	
home@	*NAME*	home@ /var/alt_mail
/	var/alt_mail/*NAME*	
var/	alt_mail/*NAME*	
alt_mail/	*NAME*	
NAME		

Table 3: Resolving `/var/mail/`*NAME* to `/var/alt_mail/`*NAME*

terminates. When hlfsd runs, it hides this symbolic link, and provides our "dynamic" symbolic link. This trick at least provides us with an alternate place to deliver mail when things go wrong, rather than bounce or drop the mail.

A cron job on our systems checks the alternate mail spool directory several times a day. Any messages found there are resent to their rightful owners. The remailing script can be run as often as needed. Each invocation of the script deals only with newly lost mail since the previous invocation; the script locks and renames the lost mailbox file to a unique name, before parsing and remailing it.

Similar to amd, hlfsd can log debugging and various status information to a designated log file or using the syslog[22] facility. The SA may choose to watch these log files and facilities and be notified when serious problems occur such as a full filesystem.

Avoiding Hangs

As described in the section on execution flow, hlfsd forks a child at any point where we suspect that an operation might hang. If, for example, the home machine of the user is down, and the filesystem on a client is hard-mounted, hlfsd will hang until the remote server is back up. Performing these operations in the background provides added reliability, an idea taken from amd.

Disk Space Problems

Hlfsd checks if the user's home directory is full or they exceeded their quota. It attempts to create and then remove a simple nonzero-length file in the user's spool directory, with the effective *uid* set to that of the user. If that fails, it instead returns back the name of the alternate spool directory as the value of the home symbolic link. Otherwise mail might be dropped or bounce.

Any success or failure state is recorded in hlfsd. It is left there for a specified number of seconds, after which the entry "times out" and a new actual backgrounded lookup is required. Otherwise, the cached result is used and no expensive fork is required. This simple caching feature of hlfsd has greatly improved its performance and reliability. Also see the section on problems.

Lock Files

An alternative design for hlfsd is to have it mount on top of the mail spool directory directly, instead of having the mail spool directory be a symbolic link to another link (home) within the HLFS which points to a real subdirectory of the user's home. With some modifications to the server, we could have made all of the user's mailbox files point to the right place, but it suffered from serious drawbacks:

- The spool directory would no longer be a regular directory. It would have to be managed by hlfsd. This would require the implementation of more NFS operations.
- The user's spool file would not be a regular file, but a symbolic link to such. Some mail programs remove that file, not checking if it's a symbolic link. Therefore the symbolic link would be removed. We would have had to change the server so that removing the symbolic link would first follow it and remove the file it was pointing to. The same goes for all operations which require access to the user's mail spool file.
- The worst problem was that different UAs and MTAs use different methods for locking the mail file. Some of them create temporary files named ${USER}.lock, others use the mktemp library call to generate unique names. Our method avoids the need to figure out all the different methods used in locking mail files, and usage of temporary files.

An alternate way to avoid the need for lock files is to deliver mail one message per file using a different system such as with INN[19] and NNTP[10]; however, this would require modifications to all UAs and MTAs.

Source Code Size, Availability, and Portability

Hlfsd is less than 2500 lines of C code, including comments and white-spaces. However, it makes use of almost 4000 lines of code from the amd distribution itself.

Hlfsd is available in source form as part of a special distribution of amd. It can be retrieved via anonymous ftp from ftp.cs.columbia.edu in the directory /pub/amd.

Hlfsd is built as part of the special distribution of amd available from our ftp server. It is almost as portable as amd is. It is only the lack of access to certain machines that stopped us from porting hlfsd to the numerous platforms amd runs on. At the writing of this paper, hlfsd has been successfully ported and running on SunOS 4.1.3, HP-UX 9.0.1, and Solaris 2.2. Those represent the 3 main system types amd runs on and span most Unix flavors: a BSD-style system, an SVR4-BSD hybrid, and a system very close to SVR4, respectively.

Evaluation

This system is implemented and has been in use on a number of machines for more than a year now. For the first nine months hlfsd was in experimental use. We have since deployed it on most production machines in our department, spanning over a 100 hosts and 3 different architectures.

The goal of this work is to expand the accessibility of electronic mail, improve overall reliability and stability of this vital service, while at the same time maintain the sanity of our SAs (yours truly included). For this to really work, it must have:

1. Very high availability.
2. Little overhead.
3. Little hassle for users and administrators as the system is being used or installed for the first time.

Performance

We have carried out some measurements to quantify the above requirements and more. The tests were performed on a Sun SPARCstation-2 running SunOS 4.1.3.

Accessing a local spool file via `stat` normally takes 180 msec without `hlfsd`. If `hlfsd` is running and has the user's entry already cached, it takes 60 msec more to access the file. This overhead is attributed to the fact that the kernel has to access a user-level NFS server, making several context switches.

If the entry is not cached, `hlfsd` forks a child to perform operations which may cause it to hang. The overhead of that `fork` and other checks is an additional 70 msec (or 130 msec over the regular lookup not using `hlfsd`). However, this overhead is incurred only once in 5 minutes, because the result of each check is cached for that long by default.

The above times are somewhat significant, but not by much, considering the use of a user-level file-server. (By comparison, in our environment it takes about 0.5 seconds to access a new filesystem using `amd`.) Given the benefits of `hlfsd`, we feel that a minimal access slowdown is a small price to pay. In practice, over 12 months of usage we have noticed no visible degradation of service.[13]

The internal data structures (tables and caches) require 50 bytes per user on the system. In our environment (750 users), that totals about 37KB – rather insignificant given that workstations these days come installed with at least 16-32MB of RAM.

Remailing Lost Mail

The `hlfsd` distribution contains a `perl`[26] script called `lostaltmail`. Remailing a single message with a body size of 1KB takes an average of 1.2 seconds (total time). In our department, resending an average mailbox file takes about 20 seconds.

Initially we ran the script once a day, but found having to wait up to 24 hours for lost mail redelivery too long. We then experimented with running `lostaltmail` once an hour. However, we found that frequency too fast. The most likely situation in which `hlfsd` will repoint its symbolic link to the alternate spool directory is when the user's filesystem is full. A full filesystem is a persistent situation that in most cases takes some time to get fixed, as it requires human intervention. If the situation that causes `hlfsd` to use the alternate spool directory is likely to persist, running the `lostaltmail` script will consume unnecessary resources, only to redeliver the mail back to the alternate spool directory. We finally settled on running `lostaltmail` between 6 and 12 times a day. Depending on the amount of lost mail expected, the script could be run more or less often.

Reliability

We have simulated worst-case scenarios by filling up a user filesystem and letting `hlfsd` decide to redirect mail to the alternate spool directory. At this point we filled up that filesystem as well. `Hlfsd` kept on pointing to the alternate spool directory during the cached entry interval, but we observed no mail lost. Instead, the sending side detected that the filesystem was full, and kept the message in the remote (private) spool directory. This is the default behavior `sendmail`[1] provides.

`Hlfsd` does not introduce any new problems; that is, if a filesystem is completely full, whatever behavior your current LDA provides is maintained. Since `hlfsd` uses both the user's filesystem and an alternate spool directory, it actually increases the availability of mail services, by "virtually" increasing the disk space available for mail spooling.

Once space has been freed in the user's filesystem, and the cached entry expired, `hlfsd` pointed its symbolic link back to the user's home directory. The next time the remailing script ran, all "lost" mail got resent to its owners.

Since the installation of `hlfsd` in our production environment, we have seen a few cases of lost mail being resent, mostly due to full filesystems. We know of no case where mail was completely lost.

Installation

Since `hlfsd` was written by SAs for other SAs, we have provided it with several command-line options to use at startup time, enabling `hlfsd` to be tailored for a particular environment. Needless to say, a man page is provided, as well as complete source code. Furthermore, we included a few scripts written in `sh` and `perl` which we use in our environment to re/start `hlfsd`, test for possible configuration anomalies, and resend "lost" mail.

The most significant work that SAs need to do is identify programs that need to access mailbox files of other users, and "*setgid*" them to HLFS_GID. In our environment we had to do that for `comsat`, `from`, `finger` and a few others. Our environment uses the `rdist`[5] automatic software distribution program, and thus these changes were required only in one place – the top of our `rdist` tree.

[13]The SAs group felt so convinced that `hlfsd` was working well, that we were the first to use it on *our* home machines.

Problems

There are a few problems, some of which cannot be easily resolved:

- Some programs need to be *setgid* to the special HLFS_GID group. There is no easy way to locate them other than knowing ahead of time what they do. Note that if the programs are not *setgid*, the only consequence is that these programs are unable to find mailboxes. However, with other methods, if $MAIL is not used, mail is not delivered.
- It is possible that the status of a home directory access will change during the time that `hlfsd` caches this information. Picking a smaller cache expiration time can alleviate this problem, but it increases the resources taken by `hlfsd` and slows down access to mail. It is left for the individual SAs to change this default value.
- Any logins with the same *uid* and a different home directory may have mail delivered or read from any of their home directory pathnames. `Hlfsd` stores pathnames in an internal hash table keyed by the *uid*; therefore, it is undefined which pathname is returned in the case of multiple users with the same *uid* and different home directories. We provide a script which checks for this situation and warns the SAs.
- On systems that cannot turn off the NFS attribute cache, the kernel might return the same symbolic link name for two different users who access the spool directory consecutively, possibly resulting in mail getting delivered to the wrong mailbox! On these systems, `hlfsd` will not run unless started with a special option. In that case it will set the attribute cache value to the shortest possible interval, but it may not be sufficient.

Related Work

The idea of dynamic or variable pathname components is not new. HP-UX does this with context-dependent files[8], and Mach with the "`@SYS`" variables[4]. Both of these implementations support replacement of pathname components by kernel variables. Apollo's DOMAIN/OS supported a more sophisticated system where arbitrary user environment variables could be referenced in pathnames[11,12]. On the issue of having a user's home files and mailbox file reside in one location, Plan 9's *attach* operator can be used to unify several file servers into one user name space[14].

What is new about our idea is that we do not require any change to any part of the filesystem implementation in the kernel. All that is required are RPC and NFS, making the system much more widely applicable.

Though at first it may appear that `amd` can do what hlfsd does, it can't. `Amd` cannot return different pathnames as a value of a symbolic link depending on who accessed it. See the earlier section on Distributed Mail Spool Service for more details on various ways in which `amd` cannot help the way `hlfsd` can.

Mail-reading Servers

The future of mail reading and sending may be similar to that used by the NNTP protocol used for managing NetNews[10,19]. That is, a special-purpose server which provides network connections for reading and writing mail remotely.

Several such programs exist, most notably IMAP[16] and POP[17]. However, use of these servers is limited at this time because most MTAs and UAs have not been converted to use them, or they require special environments (the *Andrew Message Delivery System*[18] requires AFS). Porting those applications for most popular environments is not going to be an easy task. Nevertheless, the benefits of such services over that of `hlfsd` would include faster and more reliable service, plus greatly expanded functionality (possibly providing threads information for threaded mail readers).

Conclusion

We have described the benefits of delivering mail to users' home directories, the traditional ways to do that and why we think they are inadequate, and the design, implementation, performance, and convenience of our alternative.

The main contribution of our work is the idea mail can be reliably delivered to user's home directories for easy access with very little overhead, user hassle, or the need for extensive intervention on the part of SAs.

A working prototype version of `hlfsd` was written in one weekend. However, the ideas represented in the work span several years of experience in network programming (especially RPC), NFS, `amd`, and mail systems.

Future Work

It would be possible to integrate some of `hlfsd`'s functionality into `amd`, by providing special keywords like `${home}`, `${user}`, and `${group}` for use in `amd`'s maps.

We plan on making sure `hlfsd` is as portable as `amd` is, and improving its performance as much as possible. An RPC interface for querying `hlfsd`'s status is needed as well.

Alternative Uses

`Hlfsd`'s primary use is that of a mail-spool redirector. However, it can be used to perform other tasks. All it takes are the right command-line options:

- `Hlfsd` can manage the `/var/tmp` directory. Thus every user who uses `/var/tmp` would actually be using a subdirectory within their own home directory, rather than taking from system-wide resources.
- Other types of user-specific files which get spooled to a particular host, such as *Secret Mail* [24] or electronic faxes can also be redirected for spooling into home directories.

Acknowledgments

Special thanks go to Daniel Duchamp for his invaluable comments on the paper, to James Tanis who wrote the remailing script and provided useful feedback, and to all members of the technical staff who helped in stress-testing `hlfsd`. We would like to thank the many members of the amd mailing list (`amd-workers@acl.lanl.gov`) for providing valuable discussion on the subject.

This work was supported in part by a National Science Foundation CISE Institutional Infrastructure grant, number CDA-90-24735.

As `hlfsd` uses parts of the `amd` distribution, it is distributed under the same restrictions and licenses that `amd` is.

Author Information

Erez Zadok is an MS candidate and full-time Staff Associate in the Computer Science Department at Columbia University. His primary interests include operating systems, file systems, and ways to ease system administration tasks. In May 1991, he received his B.S. in Computer Science from Columbia's School of Engineering and Applied Science. Erez came to the United States seven years ago and has lived in New York "Sin" City ever since. In his rare free time, Erez is an amateur photographer, science fiction devotee, and rock-n-roll fan. Mailing address: 500 West 120th Street, Columbia University, New York, NY 10027. Email address: ezk@cs.columbia.edu.

Alexander Dupuy has been a Senior Research Staff Associate for the Distributed Computing and Communications Lab in the Computer Science Department at Columbia University for the last 7 years. He has recently taken a position at System Management ARTS, a small startup company developing network and systems management technology. A native born and bred New Yorker, he insists that working in the suburbs is not the first step towards living there. Mailing address: System Management ARTS, 199 Main Street, Suite 900, White Plains, NY 10601. Email address: dupuy@smarts.com.

References

[1] E. Allman. SENDMAIL – An Internetwork Mail Router. In *UNIX System Manager's Manual.* University of California, Berkeley, 1986.

[2] F. C. Baran. MW: Mail-Watch. An unpublished manual page, Academic Systems Group, Columbia University Center for Computing Activities, 1987.

[3] B. Callaghan and T. Lyon. The Automounter. In *Proc. 1989 Winter USENIX Conf.*, pp. 43-51, January 1989.

[4] M. N. Condict. Configuring and Building Mach 3.0. OSF Research Institute, Grenoble, France. Unpublished notes available via ftp from `mach.cs.cmu.edu` in the file `doc/notes/kernel_build.doc`.

[5] M. A. Cooper. Overhauling Rdist for the '90s. *Large Installation System Administrators Workshop Proceedings*, pp. 1-8, USENIX, Long Beach, CA, October 19-June 23, 1992.

[6] A. J. Findlay. The Home-Directory Mail System. In *EUUG News*, Autumn 1988.

[7] J. Fulton. MIT X Consortium. X11R5 Reference Manual Pages, Section 1: "xbiff(1)", 1988.

[8] Hewlett-Packard Company. HP-UX Release 9.0 Reference Manual, Section 4: "cdf(4)", August 1992.

[9] IBM Corp. AIX Commands Reference, Volume 1, "bellmail(1)", pp. 1-84–1-87, December 1989.

[10] B. Kantor and P. Lapsley. Network News Transfer Protocol. RFC 977, February 1986; 27 p.

[11] P. J. Leach, P. H. Levine, B. P. Douros, J. A. Hamilton, D. L. Nelson, and B. L. Stumpf. *The Architecture of an Integrated Local Network.* In *IEEE Journal on Selected Areas in Communications,* **SAC-1** (5), pp. 842-856, November 1983.

[12] P. H. Levine. *The Apollo DOMAIN Distributed File System.* In *NATO ASI Series: Theory and Practice of Distributed Operating Systems,* Y. Paker, J-P. Banatre, M. Bozyiğit, pp. 241-260, editors, Springer-Verlag, 1987.

[13] J. S. Pendry and N. Williams. *Amd – The 4.4 BSD Automounter.* Imperial College of Science Technology and Medicine, London. March 1991.

[14] R. Pike, D. Presotto, K. Thompson, and H. Trickey. *Plan 9 from Bell Labs.* In *Proceedings of the Summer 1990 UKUUG Conf.*, London, July, 1990, pp. 1-9.

[15] J. B. Postel. Simple Mail Transfer Protocol. RFC 821, August 1982; 68 p.

[16] J. Rice. Interactive Mail Access Protocol. RFC 1203, February 1991; 49 p.

[17] M. Rose. Post Office Protocol. RFC 1225, May 1991; 16 p.

[18] J. Roseneberg, C. F. Everhart, and N. S. Borenstein. *An Overview of the Andrew Message System*. In *Proceedings of the ACM SIGCOMM '87 Workshop*, Stowe, Vermont, August 11-13, 1987, pp. 99-108.

[19] R. Salz. *InterNetNews: Usenet transport for Internet sites*. In *Proc. 1992 Summer USENIX Conf.*, pages 93-98, June 1992.

[20] R. Sandberg, et al. Design and Implementation of the Sun Network Filesystem. In *Proc. 1985 Summer Usenix Conf.*, pages 119-130, June 1985.

[21] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "biff(1)", September 9, 1987.

[22] Sun Microsystems, Inc. SunOS Reference Manual, Volume II, Section 3: "syslog(3)", September 9, 1987.

[23] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "comsat(8c)", and "in.comsat(8c)", September 9, 1987.

[24] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "xsend(1)", "xget(1)", and "enroll(1)", September 9, 1987.

[25] Sun Microsystems, Inc. NFS: Network File System Protocol specification. RFC 1094, 1989 March; 27 p.

[26] L. Wall and R. L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, CA (1991).

[27] R. S. Zachariassen. *ZMOG: The ZMailer Operations Guide*. Available via ftp as part of the ZMailer distribution from `ftp.uu.net` in the `networking/mail/zmailer` directory.