AFS ISSUES on FNALU

DCD Design note #62.3
Steve Hanson

1.0 Why AFS?

AFS is a modern implementation of distributed file serving. It has several advantages over older file systems such as NFS. FERMILAB is investigating the use of AFS as a distributed file service model, and it was decided that the FNALU cluster would serve as a test of this model in a production environment. Briefly, the advantages of AFS in our environment are:

1.1 Performance

AFS institutes local file caching on the client machines. A system of callbacks is implemented to keep the caches on the client machines current with the copy of the file on the server. The use of caching should increase performance both from the viewpoint of the user and by decreasing network traffic.

1.2 Consistent file organization over distributed servers

AFS provides a consistent view of the served file tree across all systems that read the file system. The layout of the file tree is determined entirely on the server end of the relationship. Therefore, logging in to any machine on the internet, the AFS file tree for FERMILAB will look the same. The organization of this tree is totally conceptual, and the actual physical placement of the files is hidden from the user. There is no need to be aware of what server holds a particular file system.

1.3 Improved Authentication of users

AFS uses a modified version of the Kerberos authentication scheme from Project Athena. This allows much higher security and more powerful authentication of users. However, it does impose some potential problems on the user, which will be covered later.

1.4 Improved Server Management

Several features and procedures in AFS provide an improvement on the traditional UNIX procedures. AFS file systems are organized out of a (typically) large number of logical volumes. Each of these volumes has a separate ACL associated with it, which allows finer setting of permissions than the normal UNIX file protection mechanism. Additional ACLS can also be placed on any directory in the AFS tree, whether it is at the top of a volume or not.

1.5 Replicated and Backup Volumes

A very useful feature of AFS is the concept of replicated volumes. In addition to a normal read-write volume, individual volumes may be replicated in another location. Typically the replicated volumes would reside on a separate file system and normally also on a separate file server machine. This allows for redundancy in the case of a system failure. The replicated volumes are read-only volumes. Thus, updating files in the read/write volume doesn't automatically update the files in the read-only replicas. The volume must be manually released to the replicas. Replication only works well on volumes that are not frequently updated. Typically these would be volumes of site-wide executables like the products directories, etc.

Additionally, AFS has a concept of backup volumes. These backup volumes are similar to read-only replicas. They don't take up any file space, but are mirrors of the original volume. They only keep track of copies of files that have changed in the original volume. Thus, the backup volume is a snapshot in time of what the file system looked like when the backup was created. These volumes are used either to back up the file system (the backup volumes are made, and are then backed up, presenting an exact snapshot of what the file system looked like when the volumes were created) and to create backup volumes for users. Thus the user could always go to a backup of, for example, their home directory volume from the day before and retrieve a file. We have not implemented backup volumes for this purpose on FNALU, but may do so in the future.

1.6 Server function redundancy

AFS also allows for a fair amount of server process redundancy, so that if any individual server goes out, the entire cell will remain running. We have not been able to implement this yet as we only have two AFS licenses and a minimum of 3 servers is required to have redundant processes. Therefore, in the current configuration, if fibs01 should go down, the entire AFS cell will become unavailable. A third license is being aquired, so it is hoped that the cell will become fully redundant some time in early August.

2.0 User Issues Presented by AFS

A number of user education issues are raised by AFS, as well as some actual problems that will be caused for certain classes of users. This paper will attempt to address some of these issues and provide some solutions where they are known.

2.0.1 Command formats

The various AFS commands have a (reasonably) consistent format. There are several commands that the average AFS user may need from time to time. Each of the major AFS commands deals with one of the subsystems (actually, one of the server daemons) in AFS. The primary commands are:

`fs` - which is the file system command. This command is used to deal with file system issues, such as checking permissions on a directory or volume, checking quota, making mount points, finding where a volume is mounted in the file tree, etc. To obtain help on using fs, you can simply type

```
fs help
```

basically this same help mechanism works on all of the AFS commands (well, most of them, anyway). To get help for a particular fs command (for example, fs listacl) you may type something like

```
fs listacl -help
```

`kas` - the Kerberos authentication server - This is used to query or interact with the Kerberos system - typically this would be to check tickets granted, set a user password, etc. This is less likely to be of a lot of use to the typical user.

`pts` - The protection server. This is used for things like checking group membership, adding a member to a group, adding a user to the protection server, etc.

`vos` - The volume server. This is primarily used by the administrator. It is used to look at the system volumes, move volumes, etc. It isn't usually available on client machines.

`klog` - Used to obtain kerberos authentication. Normally users on this system won't need to klog as they will be authenticated by the Kerberized login program.

`knfs` - Used to enable authenticated access from an NFS client mounting the AFS tree through the NFS translator. This is only likely too be useful to people who don't have AFS installed on their machines. Currently this is going to be the only way to access the AFS files from an SGI machine, for example. It requires logging in to the NFS translator (currently fibi01) and authenticating there. Then upon returning to the NFS client the user will be authenticated to access files. The NFS security is considerably more lax as a practical matter.

`tokens` - Used to check the tokens the user is holding.

`unlog` - used to discard all tokens. It is best to unlog before logging out.

I have included a brief reference to the AFS commands at the end of this document.

2.1 Authentication Issues

On the fibi01 IBM interactive system users are granted Kerberos authentication automatically at the time they log in. Providing a user name and matching password is considered proof of the user's identity. Because a modified login procedure is used on these machines, the process of logging in provides the user with an AFS ticket that identifies the user as legitimate. This authentication is passed on to all processes that are spawned from the login process. Thus, any normal sort of operation that will be done interactively by the user will be authenticated, and will have access to files in the AFS tree that have permission turned on for the login name of the user.

Tickets that are granted by the Kerberos server have a limited lifetime. Currently the default tickets granted by AFS are good for 24 hours. Thus, the user will have access to files on the system for 24 hours after logging in. This policy may change, but for the time being the implication for the user is that file permissions will mysteriously disappear after 24 hours. This will obviously cause some problems for long-running processes, batch processes, at jobs, and some other classes of work. We will be exploring solutions to these problems, which we hope will be more creative than simply granting extremely long-lived tickets. Two current solutions are being investigated. The FNCLUB system is using a program called fbatch, which allows batch jobs running on fnclub to access AFS files by NFS authenticating processes on fnclub, and then accessing the files through the NFS translator. Natively on FNALU, experimental support is being given to batch processes running under the LoadLeveler product from IBM, which understands AFS and provides authentication to processes automatically. We will additionally be experimenting with several other solutions such as the reauth daemon, which perioidically reauthenticates a running process.

There are several things worth noting about the process of obtaining a token. One of the confusing aspects here is that users may potentially have *two* different passwords - one for the AFS/Kerberos system, and one for the traditional UNIX login. In general we are setting up users on fibi01 to primarily use their Kerberos password. The thing to look out for here is that changing your password with the normal *passwd* command will change your UNIX password, which is probably not what you want. If you then log in with the changed password, you will be authenticated in UNIX but not in AFS. Thus, although you will be logged in, you will probably not be able to access your own files. The normal procedure for changing your password on fibi01 will be to use the *kpasswd* command, which will change the Kerberos password only.

Finally - logging out doesn't destroy your Kerberos token. Your token will live for the default lifetime. Before logging out, you should issue the unlog command. It may be helpful to put unlog into the logout script of whatever shell you are using.

To see what token or tokens you currently hold, you may use the *tokens* command. Commands are covered in detail later in this paper.

Details of the authentication process are covered in the *AFS User's Guide*.

One practical issue raised by the Kerberos envirronment involves the use of the Berkeley networking programs such as rlogin, rsh, and rcp. Normally systems are *equivalenced* to enable the use of these protocols. The equivalencing of the machines implies that once a user is logged into one system they may login to the equivalenced machines without providing further proof of identity, such as a password. This won't fit in with the Kerberos authentication system. We have installed token-passing versions of a lot of software on the FNALU IBM systems in an attempt to make life less painful, but they have at teh same time changed the behavior of these programs from what might be expected.

Token-passing is available for rsh and rcp. The general effect of this is that if a user has authenticated into the **`fnal`** cell, it is possible to use rcp and rsh more or less normally between machines - the token will be passed along with the request and the network communications will be authenticated. Thus the user can rcp or rsh and be authenticated on the remote machine automatically. This is set up with the usual UNIX method of using /etc/hosts.equiv or .rhosts to equivalence systems.

However, the issue with rlogin is more complicated. When equivalencing between two machines is enabled, the rlogin protocol will not ask for a password. This is very convenient on normal UNIX systems. However, in AFS systems, the lack of a provided password makes the remote user untrusted. Thus, if two systems are equivalenced, rlogin will work and no password will be requested, but the user will **`not`** be authenticated at login time. Therefore, the choices are either to use telnet instead, which will automatically authenticate the user, or to use rlogin and then to immediately authenticate using pagsh and klog.

Likewise, certain other programs in the normal FERMI environment are likely to break in this environment. One known program is zftp. It should not be too difficult to convert these programs to do authentication through the AFS libraries rather than the normal UNIX authentication routines, but unfortunately these libraries are currently not documented by Transarc. We hope this will be resolved some time before June, '93.

One final note - There is also an AFS-NFS translator available. The entire FERMI AFS tree is exported from fibi01 as an AFS mountable file system. Thus, the AFS files may be mounted on any system that works as an NFS client (supposedly including VMS systems that have NFS client software). However, anyone accessing the files will essentially be granted permissions as system:anyuser (that is, only files that are readable by system:anyuser will be readable). Users may authenticate with the AFS/NFS translator by logging into fibi01 and using the **`knfs`** command. Typical use would be as follows:

```
knfs pogo.fnal.gov 1231
```

assuming that the NFS client is node pogo and the users UID is 1231. The NFS authentication is pretty weak in some ways. The NFS translator may be removed at some time in the future, but it does currently provide a path for accessing AFS files on those machines that don't have AFS software available, such as SGI machines.

Currently the SGI machines and the fibb01 IBM batch machhine access the AFS tree through the translator. A set of shell scripts has been written that make authentication less painful in this environment. They simply replicate the actions of the normal AFS authentication programs (such as klog, unlog, tokens, etc) by doing the appropriate NFS authentication actions on the file server. Thus a user can log in to one of the SGI systems and use the klog command on the SGI, which will authenticate the user to the cell and set up NFS permissions to the user on the client system. It is hoped that as soon as the SGI machines are capable of being native AFS systems and we manage to find better batch processing solutions, that all of the FNALU machines will access AFS without benefit of the translator, which will simplify life considerably.

2.2 File Permissions

This is a confusing issue, so try to follow along. File protection in AFS is quite different than in a normal UNIX file system. A normal UNIX file or directory has nine major mode bits. They grant read, write, and execute permission for the owner, group, and world users respectively. This is shown in the output from ls -l as follows:

```
d-rw-r--r-- 1 hanson 1222 Oct 24 1991 README

-rwxr--r-- 1 hanson 311 Oct 31 1991 doit

-rw-r--r-- 1 hanson 1040384 Oct 31 1991 patchtar

drwxrwxr-x 2 hanson 2048 Oct 31 1991 shells

drwxrwxr-x 4 hanson 2048 Oct 24 1991 tree
```

a. First field is the type of the element (- for file, d for directory, l for symbolic link, etc.)

b. The next three fields list the "user" read, write and execute permissions.

c. The three fields after that list the read write and execute permissions pertaining to the group owning the file.

d. Finally, the last 3 fields list the permissions for all other users on the system.

In AFS, you can use the standard chmod command just as you would in a standard UNIX file system. However, the behavior of the file will be different. Although all of the bits may be changed or examined on a file in the AFS file system, only the owner bits are actually used in AFS. They apply to *all* users of the file. Thus to turn off write access to a particular file, you will need to turn off the *owner* write bit of the file. This will inhibit writing to that file for all users, including the owner. In the current AFS implementation, file level permissions are only changeable for the owner bits.

Other permissions are done with ACL's, which take effect on the directory level only. ACL's can be set for various users and groups. The permissions granted in the ACL are:

2.2.1 l - lookup rights

Lookup permissions allow the possessor to

- Issue an ls command on files in the directory
- Examine the ACL for the directory
- Access the directory's own subdirectories, which are protected by their own ACL's

2.2.2 i - insert rights

Insert permissions allow the user to

- add new files to the directory, either by creating new files or by copying files in.

2.2.3 d - delete rights

Delete permissions allow the user to

- remove files or move them to other directories

2.2.4 a - administer rights

Administer permissions allow the holder to

- change the ACL for a directory. Users always have this right for their home directories, even if they accidentally remove this acl themselves.

2.2.5 r - read rights

Read rights allow the holder to

- look at the contents of the directory. For files in this directory the possessor can read the data in the file.

2.2.6 w - write rights

Write rights allow the holder to

- Modify the contents of the files in the directory and to change their UNIX mode bits with chmod.

2.2.7 k - lock rights

Lock rights allow the user to

- run programs that need to "flock" files in this directory.

Rights may also be referred to by special names that refer to commonly-assigned combinations of rights.

- Write - all rights except administer (that is, *rlidwk* rights)
- read - read and lookup only
- all - all seven rights (*rlidwka*)
- none - no rights - this removes the user's entry from the ACL entirely.

Users may examine the ACL setting on a particular directory with the

fs listacl

command. Rights are set with the

fs setacl

command.

Please note that a reasonable set of permissions will be set up on each volume by the system administrators when the volume is created. In general users will be granted *all* permissions on their home directories. ACL's are by default inherited by child directories when they are created.

Additionally, home directories will be created with the permissions of system:authuser set to rl. System:authuser is a special system user name that is assigned to any user who is authenticated on the server. Thus, by default any files in your home directory and directories within your directory will be readable by any other user who has authenticated with our AFS server (including, potentially, offsite users who have accounts with us). If you want to read-protect directories within your home directory, or your home directory itself, you will have to do so by removing the ACL for system:authuser. Note that if you turn off rl permissilon to your home directory for system:anyuser, you will find that the system can no longer read your .forward file, etc. We are working on moving toward a more sophisticated model in which your home directory can be protected more stringently, either by having a public directory under your home directory, into which things like .forward may be simlinked, or by using an authenticated mail system such as AMDS. There will probably be some resolution of this in an August, 1993 time frame.

2.3 Quota issues

Currently we are creating home directories on the FNALU systems with 10 Mbyte quotas. Quota change requests must go through the ACCESS office. In the future quota will be set according to the size that is approved when a project petitions for use of FNALU. You can check quota on a volume by using the

fs listquota

command. Likewise, quota may be set on a volume (if you have permission to set quota on the volume) with the

fs setquota

command.

3.0 AFS in the FUE/PRODUCTS/UPS world

I will give here a few practical examples of how AFS permissions have been implemented in the products area on FNALU to show how the file permissions might work in a real-world situation we are familiar with. This will also serve as a guide for those people who are maintaining UPS products on the FNALU cluster.

3.1 File layout

The layout of the files in the products area is the same as it is on other multi-architecture clusters at the lab, with the exception of the fact that the products tree begins at

/afs/fnal/products

rather than /usr/products. If you are unfamiliar with the multi-architecture layout, it is very similar to the normal single architecture layout of files except that each separate platform has a directory under the products directory which holds the actual products code. The database directory is shared among platforms. Thus, there are directories for

/afs/fnal/products/AIX

/afs/fnal/products/IRIX

/afs/fnal/products/SunOS

etc. The only one implemented currently is the AIX directory, but the others will follow shortly.

A design decision was made to allow product developers to work within individual products directories logged in as themselves. Thus, each of the products has a separate AFS volume. The ACL for each volume has been set so that the product developers have full permissions on the volume. Any authenticated user on the system may read the contents of the volume, as well as do lookups. Products also has full permissions on the directory. Thus, as an example, the permissions on /afs/fnal/products/AIX/bufio are set as shown:

pogo% fs listacl bufio

Access list for bufio is

Normal rights:

```
 products:bufio rlidwka
 system:administrators rlidwka
 system:authuser rl
 products rlidwka
```

The quotas on each directory are set approximately to allow three revisions of the product and to allow some extra space. On the same directory, the quota is shown as:

```
pogo% fs listquota bufio
Volume Name Quota Used % Used Partition
products.AIX.bufio 3500 872 25% 13%
```

To be able to declare products the installer must be able to write into the **/afs/fnal/products/ups_database** directory. The permissions on this directory are set so that all permissions are granted to a group called products:upsdatabase -

```
pogo% fs listacl ups_database
Access list for ups_database is
Normal rights:
 products:upsdatabase rlidwka
 system:administrators rlidwka
 system:authuser rl
 products rlidwka
```

The products:upsdatabase group has as its members ALL of the products maintainers. Thus any of the products maintainers may make a product current.

```
pogo% pts membership products:upsdatabase
Members of products:upsdatabase (id: -288) are:
 edel
 kreymer
```

```
lebrun
trang
janetc
b92799
marafino
garren
jeffk
sandrac
stein
```

4.0 Currently Open Issues and Problems

4.1 File System Permissions are Too Lax

This was mentioned above. We hope to have a more sophisticated scheme in place to allow home directories to not be world readable some time in August ‘93.

4.2 Backup Issues

There have been some issues with creating authenticated backups in the AFS file system, particularly with doing the backups in an automated fashion. The groundwork has been done for this. Backups are now done nightly, but require the system administrators to load the tapes. CPS has just been installed, and the XOPER system will shortly be used to request tape mounts for backup tapes and other tape mounts automatically. This should all be operating by August 1, 1993.

4.3 Mounting of User-Accessible Backups

We have recently made user-accessible backups available. This function has not been heavily tested, but the backup volumes are currently available for all home directories. The backup volumes are mounted at /afs/fnal/files/backup, in a tree that simulates the full AFS tree. Since this has been done, users can restore files themselves that have been accidentally deleted or damaged since the last backup run.

4.4 User Documentation

There is not a large availability of user documentation for AFS. Many of the online manual pages are somewhat out of date, or are confusingly named. We have obtained a set of user-written manual pages. We are looking at merging the different sets of documentation. We are also trying to receive a price quote from Transarc for a large volume of the AFS reference cards, which are useful as an inexpensive desktop reference.

4.5 Systools utilities

We are currently trying to make some of the DCD utilities more AFS aware. In particular, the adduser family of utilities is being rewritten for use in an AFS environment. Adduser is nearly finished, which will allow a non-system-administrator to add accounts to the FNALU cluster, so it can be handled like the other major clusters on site. This function should be available by July 24, 1993.

4.6 Availability of AFS on the SGI platform

Currently we can support AFS on Solaris 1, and AIX 3.2 RS/6000 platforms. We are attempting to purchase an AFS license for Solaris 2, which should be available later this fiscal year. However, there is currently no supported AFS software for SGI. We have received an ALPHA release of the AFS client (only) for SGI. However, it requires IRIX 5.01 to run, which we are not currently supporting anywhere. It seems likely that we will not be in a position to support SGI AFS as a matter of course until around the end of this calendar year, and that AFS serving may not be running successfully even that soon. This has slowed down life considerably, as the 8 processor SGI system in the cluster was purchased primarily as a batch system and AFS server. Most of our disk is therefore not available for AFS serving. There is also a problem in that we do not have copies of some software to run under 5.01 IRIX, such as the LoadLeveler software. Therefore, we may be faced with having to make a choice between using the SGI as an AFS server or as a batch engine. It may be necessary to move some of the peripherals from fsgm01 to one of the AFS file servers to accomodate growth in AFS file serving.

5.0 Oddities of the AFS file system

There are a few minor issues with AFS that are not immediately apparent and that should be explained so that if you run across them you won't be too shocked or confused.

5.1 File ownership is almost meaningless

Since the ownership bits on file permissions apply to ALL users who have access allowed by the ACL, the ownership of the file is nearly unimportant. Programs which specifically test for ownerhip of a file will of course care about this. But the ownership of a file means nothing as far as file *access* is concerned.

5.2 Hard links are only legal within a volume

This may cause some problems with some Makefiles, etc.. Usually doing a symbolic link instead is an acceptable workaround.

5.3 Problems with find, etc.

You should be *very careful* about anything that traverses the file system on a machine that has /afs mounted. Please be aware that a find on your system starting at / will traverse the *whole* AFS file tree - including all the other sites that we have mounted on our cell. This is particuarly a problem on some workstations like Suns that by default run a nightly cron job that traverses the whole file system. Also note that using things like pruning NFS mounted file systems in your find won't help - find can't tell the difference between local files and AFS files. I am looking into obtaining a version of GNU find that is AFS aware so that we will have a more reasonable way of pruning finds on systems.

5.4 User confusion over file permissions

Just to bring this up one more time. Users who expect to have the file system permissions on AFS files act anything like they do in a normal UNIX file system are going to be mighty disappointed or confused. The ACL on a directory is the overall governing permission on a flie. The *only* effect of the UNIX file permissions is to serve as a mask where the owner permission bits will affect the permssions of all users who are granted permission by the ACL.

5.5 Batch, at, and cron processes

Users should be aware of the fact that there will be problems with running programs with batch, at, cron, or any other program that will spawn a job that isn't part of the user's login process group. You can certainly schedul a job to run with at or cron, but the job won't run with authentication, so it will likely not be able to write output anywhere in the AFS file space (or even read data in some cases). We are working on batching solutions for users that will work in this environment, but the typical UNIX utilities will have problems. Likewise, be aware of the fact that being root on your computer grants you no special permissions in AFS file space, and in fact will be the same as being any other user. There is no such thing as being "authenticated as root".