mount-info `restarted fstype ${type} on ${fs}`
Amd is using a pre-mounted filesystem to satisfy a mount request.

mount-info `unmounted fstype ${type} from ${fs}`
A file system has been unmounted.

mount-info `unmounted fstype ${type} from ${fs} link ${fs}/${sublink}`
A file system of which only a sub-directory was in use has been unmounted.

Acknowledgements & Trademarks

Thanks to the Formal Methods Group at Imperial College for suffering patiently while amd was being developed on their machines.

Thanks to the many people who have helped with the development of amd especially Piete Brooks at the Cambridge University Computing Lab for many hours of testing, experimentation and discussion.

References

[1] B. Callaghan and T. Lyon, "The Automounter," in *Usenix Conference Proceedings,* San Diego, California, pp. 43–51, Usenix Association, January 1989.

[2] Sun Microsystems, "Automount," in *SunOS Reference Manual*, ch. 8, pp. 1583–1585, Mountain View, California: Sun Microsystems, Inc, first ed., May 1988.

[3] Sun Microsystems, "Network File System: Version 2 Protocol Specification," in *Network Programming*, ch. 7, pp. 165–185, Mountain View, California: Sun Microsystems, Inc, first ed., May 1988.

[4] Sun Microsystems, "Remote Procedure Calls: Protocol Specification," in *Network Programming*, ch. 6, pp. 143–163, Mountain View, California: Sun Microsystems, Inc, first ed., May 1988.

[5] J. Postel, "Internet Control Message Protocol," RFC 792, SRI Network Information Center, Menlo Park, California, September 1981.

[6] Sun Microsystems, "The Sun YP Service," in *System & Network Administration*, ch. 14, pp. 349–371, Mountain View, California: Sun Microsystems, Inc, first ed., May 1988.

[7] S. P. Dyer, "The *Hesiod* Name Server," in *Usenix Conference Proceedings,* Dallas, Texas, pp. 183–189, Usenix Association, February 1988.

[8] J. Postel, "Internet Protocol," RFC 791, SRI Network Information Center, Menlo Park, California, September 1981.

[9] S. J. Leffler *et al.*, *The Design and Implementation of the 4.3BSD UNIX Operating System*, ch. 7, pp. 187–223. Addison-Wesley, 1989.

[10] M. Greenwald and J. V. Sciver, "Remote Virtual Disk Protocol Specification," tech. rep., Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986.

`unable to free rpc arguments in nfs_program_1`
: The incoming arguments to the NFS server could not be free'ed.

`unable to register (AMQ_PROGRAM, AMQ_VERSION, udp)`
: The AMQ server could not be registered with the local portmapper or the internal RPC dispatcher.

`unable to register (NFS_PROGRAM, NFS_VERSION, 0)`
: The NFS server could not be registered with the internal RPC dispatcher.

7.1.2 Info messages

Amd generates information messages to record state changes. These messages are selected by `-x info` on the command line. When **syslog**(3) is being used, they are logged with level `LOG_INFO`. The messages listed below can be generated and are in a format suitable for simple statistical analysis. "*mount-info*" is the string that is displayed by amq in its mount information column and placed in the system mount table.

`mount of "${path}" on ${fs} timed out`
: Attempts to mount a filesystem for the given automount point have failed to complete within 30 seconds.

`"${path}" forcibly timed out`
: An automount point has been timed out by the amq command.

`restarting` *mount-info* `on ${fs}`
: A pre-mounted file system has been noted.

`"${path}" has timed out`
: No access to the automount point has been made within the timeout period.

`file server ${rhost} is down - timeout of "${path}" ignored`
: An automount point has timed out, but the corresponding file server is known to be down. This message is only produced once for each mount point for which the server is down.

`Re-synchronizing cache for map ${map}`
: The named map has been modified and the internal cache is being re-synchronized.

`Filehandle denied for "${rhost}:${rfs}"`
: The mount daemon refused to return a file handle for the requested filesystem.

`Filehandle error for "${rhost}:${rfs}":` *description*
: The mount daemon gave some other error for the requested filesystem.

`file server ${rhost} type nfs starts up`
: A new NFS file server has been referenced and is known to be up.

`file server ${rhost} type nfs starts down`
: A new NFS file server has been referenced and is known to be down.

`file server ${rhost} type nfs is up`
: An NFS file server that was previously down is now up.

`file server ${rhost} type nfs is down`
: An NFS file server that was previously up is now down.

`Finishing with status` *exit-status*
: Amd is about to exit with the given exit status.

mount-info `mounted fstype ${type} on ${fs}`
: A new file system has been mounted.

with level `LOG_FATAL`. Even if `amd` continues to operate it is likely to remain in a precarious state and should be restarted at the earliest opportunity.

`Attempting to inherit not-a-filesystem`
: The prototype mount point created during a filesystem restart did not contain a reference to the restarted filesystem. This erorr "should never happen".

`Can't bind to domain "`*yp-domain*`"`
: A specific YP domain was requested on the command line, but no server for that domain is available on the local net.

`Can't determine IP address of this host (`*hostname*`)`
: When `amd` starts it determines its own IP address. If this lookup fails then `amd` cannot continue. The hostname it looks up is that obtained returned by **gethostname**(2) system call.

`Can't find root file handle for` *automount point*
: `Amd` creates its own file handles for the automount points. When it mounts itself as a server, it must pass these file handles to the local kernel. If the filehandle is not obtainable the mount point is ignored. This error "should never happen".

`Must be root to mount filesystems (euid =` *euid*`)`
: To prevent embarrassment, `amd` makes sure it has appropriate system privileges. This amounts to having an euid of 0. The check is made after argument processing complete to give non-root users a chance to access the "-v" option.

`No work to do - quitting`
: No automount points were given on the command line and so there is no work to do.

`Out of memory in realloc`
: While attempting to realloc some memory, the memory space available to `amd` was exhausted. This is an unrecoverable error.

`Out of memory`
: While attempting to malloc some memory, the memory space available to `amd` was exhausted. This is an unrecoverable error.

`cannot create rpc/udp service`
: Either the NFS or AMQ endpoint could not be created.

`gethostname:` *description*
: The **gethostname**(2) system call failed during startup.

`host name is not set`
: The **gethostname**(2) system call returned a zero length host name. This can happen if `amd` is started in single user mode just after booting the system.

`ifs_match called!`
: An internal error occurred while restarting a pre-mounted filesystem. This error "should never happen".

`mount_afs:` *description*
: An error occured while `amd` was mounting itself.

`run_rpc failed`
: Somehow the main NFS server loop failed. This error "should never happen".

`unable to free rpc arguments in amqprog_1`
: The incoming arguments to the AMQ server could not be free'ed.

in the sub-directory `bin/sun4` of the filesystem `/usr/r+d`. When it was accessed a symbolic link pointing to `/a/gould/usr/r+d/bin/sun4` would be returned.

```
/defaults    type:=nfs;opts:=rw,grpid,nosuid,intr,soft
wp           -opts:=rw,grpid,nosuid;rhost:=charm \
             host==charm;type:=link;fs:=/usr/local/wp \
             host!=charm;type:=nfs;rfs:=/vol/wp
...
#
src          -opts:=rw,grpid,nosuid;rhost:=charm \
             host==charm;type:=link;fs:=/usr/src \
             host!=charm;type:=nfs;rfs:=/vol/src
#
r+d          type:=auto;fs:=${map};pref:=r+d/
# per architecture bin,etc,lib&ucb...
r+d/bin      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/etc      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/include  rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/lib      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/man      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/src      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/ucb      rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
# hades pictures
pictures     -opts:=rw,grpid,nosuid;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/pictures \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=pictures
# hades tools
hades        -opts:=rw,grpid,nosuid;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/hades \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=hades
# bsd tools for hp.
bsd          -opts:=rw,grpid,nosuid;arch==hp9000;rhost:=thpfs \
             host==thpfs;type:=link;fs:=/nbsd/bsd \
             host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=bsd
```

7 Internals

7.1 Log Messages

In the following sections a brief explanation is given of some of the log messages made by amd. Where the message is in `typewriter` font, it corresponds exactly to the message produced by amd. Words in *italic* are replaced by an appropriate string. Variables, ${*var*}, indicate that the value of the appropriate variable is output.

Log messages are either sent direct to a file, or logged via the **syslog**(3) mechanism. Messages are logged with facility `LOG_DAEMON` when using **syslog**(3). In either case, entries in the file are of the form:

date-string hostname amd[*pid*] *message*

7.1.1 Fatal errors

Amd attempts to deal with unusual events. Whenever it is not possible to deal with such an error, Amd will log an appropriate message and, if it cannot possibly continue, will either exit or abort. These messages are selected by `-x fatal` on the command line. When **syslog**(3) is being used, they are logged

```
tex             type:=auto;fs:=${map};pref:=${key}/
tex/fonts       host!=fserver;type:=nfs \
                host==fserver;fs:=/usr/local
tex/lib         host!=fserver;type:=nfs \
                host==fserver;fs:=/usr/local
tex/bin         -sublink:=${key}/${arch} host!=fserver;type:=nfs \
                host:=fserver;fs:=/usr/local
```

When `/vol/tex/bin` is referenced, the current machine architecture is automatically appended to the path by the `${sublink}` variable. This means that users can have `/vol/tex/bin` in their `PATH` without concern for architecture dependencies.

6.5 Wildcard names & Replicated Servers

By using the wildcard facility, `amd` can *overlay* an existing directory with additional entries. The system files are usually mounted under `/usr`. If instead `amd` is mounted on `/usr`, additional names can be overlayed to augment or replace names in the "master" `/usr`. A map to do this would have the form:

```
local       type:=auto;fs:=local-map
share       type:=auto;fs:=share-map
*           -type:=nfs;rfs:=/export/exec/${arch};sublink:="${key}" \
                rhost:=fserv1  rhost:=fserv2  rhost:=fserv3
```

Note that the assignment to `${sublink}` is surrounded by double quotes to prevent the incoming key from causing the map to be misinterpreted. This map has the effect of directing any access to `/usr/local` or `/usr/share` to another automount point. In this example, it is assumed that the `/usr` files are replicated on three fileservers: `fserv1`, `fserv2` and `fserv3`. For any references other than to `local` and `share` one of the servers is used and a symbolic link to `${autodir}/${rhost}/export/exec/${arch}/`*whatever* is returned once an appropriate filesystem has been mounted.

6.6 `rwho` servers

The `/usr/spool/rwho` directory is a good candidate for automounting. For efficiency reasons it is best to capture the rwho data on a small number of machines and then mount that information onto a large number of clients. The data written into the rwho files is byte order dependent so only servers with the correct byte ordering can be used by a client:

```
/defaults           type:=nfs
usr/spool/rwho      -byte==little;rfs:=/usr/spool/rwho \
                        rhost:=vaxA  rhost:=vaxB \
                    || -rfs:=/usr/spool/rwho \
                        rhost:=sun4  rhost:=hp300
```

6.7 `/vol`

`/vol` is used as a catch-all for volumes which do not have other conventional names.

Below is part of the `/vol` map for the domain `doc.ic.ac.uk`. The `r+d` tree is used for new or experimental software that needs to be available everywhere without installing it on all the fileservers. Users wishing to try out the new software then simply include `/vol/r+d/bin,ucb` in their path.

The main tree resides on one host `gould.doc.ic.ac.uk`, which has different `bin`, `etc`, `lib` and `ucb` sub-directories for each machine architecture. For example, `/vol/r+d/bin` for a Sun-4 would be stored

```
phjk        fs:=/home/toytown/ai/phjk
sjv         fs:=/home/ganymede/sjv
```

Whenever a login name is accessed in `/homes` a symbolic link appears pointing to the real location of that user's home directory. In this example, `/homes/jsp` would appear to be a symbolic link pointing to `/home/charm/jsp`. Of course, `/home` would also be an automount point.

This system causes an extra level of symbolic links to be used. Although that turns out to be relatively inexpensive, an alternative is to directly mount the required filesystems in the `/homes` map. The required map is simple, but long, and its creation best automated. The entry for `jsp` could be:

```
jsp     -sublink:=${key};rfs:=/home/charm \
        host==charm;type:=ufs;dev:=/dev/xd0g \
        host!=charm;type:=nfs;rhost:=charm
```

This map can become quite big if it contains a large number of entries. By combining two other features of amd it can be greatly simplified. First the UFS partitions should be mounted under the control of `/etc/fstab`, taking care that they are mounted in the same place that `amd` would have automounted them, *ie* `/a/`*host*`/home/`*host*. In this case `/etc/fstab` on host `charm` would have a line

```
/dev/xy0g /a/charm/home/charm 4.2 rw,nosuid,grpid 1 5
```

The map can then be changed to:

```
/defaults   type:=nfs;sublink:=${key};opts:=rw,intr,nosuid,grpid
jsp         rhost:=charm;rfs:=/home/charm
njw         rhost:=dylan;rfs:=/home/dylan/dk5
...
phjk        rhost:=toytown;rfs:=/home/toytown;sublink:=ai/${key}
sjv         rhost:=ganymede;rfs:=/home/ganymede
```

This map operates as usual on a remote machine (ie `${host}` $\neq$ `${rhost}`). On the machine where the filesystem is stored (ie `${host}` = `${rhost}`), amd will construct a local filesystem mount point which corresponds to the name of the locally mounted UFS partition. If amd is started with the "-r" option then instead of attempting an NFS mount, amd will simply inherit the UFS mount (§4.9, p16).

6.4 Architecture Sharing

Often a filesystem will be shared by machines of different architectures. Separate trees can be maintained for the executable images for each architecture, but it may be more convenient to have a shared tree, with distinct subdirectories.

A shared tree might have the following structure on the fileserver (called `fserver` in the example):

```
local/tex
local/tex/fonts
local/tex/lib
local/tex/bin
local/tex/bin/sun3
local/tex/bin/sun4
local/tex/bin/hp9000
...
```

In this example, the subdirectories of `local/tex/bin` should be hidden when accessed via the automount point (conventionally `/vol`). A mount-map for `/vol` to achieve this would look like:

```
/defaults     rfs:=/vol;sublink:=${key};rhost:=fserver;type:=link
```

```
...

#
localhost       type:=link;fs:=${host}
...
#
# dylan has two user disks so have a
# top directory in which to mount them.
#
dylan           type:=auto;fs:=${map};pref:=${key}/
#
dylan/dk2       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/2s0
#
dylan/dk5       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/5s0
...
#
toytown         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xy1g
...
#
zebedee         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/dsk/1s0
#
# Just for access...
#
gould           type:=auto;fs:=${map};pref:=${key}/
gould/staff     host!=gould;type:=nfs;rhost:=gould;rfs:=/home/${key}
#
gummo           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key}
...
```

This map is shared by most of the machines listed so on those systems any of the user disks is accessible via a consistent name. Amd is started with the following command

```
amd /home amd.home
```

Note that when mounting a remote filesystem, the *automounted* mount point is referenced, so that the filesystem will be mounted if it is not yet (at the time the remote mountd obtains the file handle).

6.3 Home Directories

One convention for home directories is to locate them in `/homes` so user `jsp`'s home directory is `/homes/jsp`. With more than a single fileserver it is convenient to spread user files across several machines. All that is required is a mount-map which converts login names to an automounted directory.

Such a map might be started by the command:

```
amd /homes amd.homes
```

where the map `amd.homes` contained the entries:

```
/defaults   type:=link   # All the entries are of type:=link
jsp         fs:=/home/charm/jsp
njw         fs:=/home/dylan/dk5/njw
...
```

```
case "$LOGFILE" in
*/*)
        mv "$LOGFILE" "$LOGFILE"-
        > "$LOGFILE"
        ;;
syslog)
        : nothing
        ;;
esac

cd /usr/local/etc
#
# -r            restart
# -d dmn        local domain
# -w wait       wait between unmount attempts
# -l log        logfile or "syslog"
#
eval ./amd -r $dmn -w 240 -l "$LOGFILE" \
        /homes amd.homes -cache:=inc \
        /home amd.home -cache:=inc \
        /vol amd.vol -cache:=inc \
        /n amd.net -cache:=inc
```

If the list of automount points and maps is contained in a file or YP map it is easily incorporated onto the command line:

```
...
eval ./amd -r $dmn -w 240 -l "$LOGFILE" `ypcat -k auto.master`
```

6.2 User Filesystems

With more than one fileserver, the directories most frequently cross-mounted are those containing user home directories. A common convention used at Imperial College is to mount the user disks under `/home/`*machine*. Typically, the `/etc/fstab` file[6] contained a long list of entries such as:

machine`:/home/`*machine* `/home/`*machine* `nfs` ...

for each fileserver on the network.

There are numerous problems with this system. The mount list can become quite large and some of the machines may be down when a system is booted. When a new fileserver is installed, `/etc/fstab` must be updated on every machine, the mount directory created and the filesystem mounted. In research environments most people use the same few workstations, but it is convenient to go to a colleague's machine and access your own files. When a server goes down, it can cause a process on a client machine to hang. By minimising the mounted filesystems to only include those actively being used, there is less chance that a filesystem will be mounted when a server goes down.

The following is a short extract from a map taken from a research fileserver at Imperial College. Note the entry for `localhost` which is used for users such as the operator (`opr`) who have a home directory on most machine as `/home/localhost/opr`.

```
/defaults       opts:=rw,intr,grpid,nosuid
charm           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xd0g
#
```

[6] `/etc/checklist` on System V

When the reference count is zero the filesystem is not mounted but the mount point and server information is still being maintained by amd.

Amq generally applies an operation, specified by a single letter option, to a list of mount points. The default operation is to obtain statistics about each mount point. This is similar to the output shown above but includes information about the number and type of accesses to each mount point.

By default the local host is used. In an HP-UX cluster the root server is used since that is the only place in the cluster where amd will be running. To query amd on another host the "-h" option should be used.

The "-u" option causes the time-to-live interval of the named mount points to be expired, thus causing an unmount attempt. This is the only safe way to unmount an automounted filesystem. It is not possible to unmount a filesystem which has been mounted with the notimeout flag.

The "-s" option displays global statistics. If any other options are specified or any filesystems named then this option is ignored.

The "-f" option causes amd to flush the internal mount map cache. This is useful for YP and Hesiod maps since amd will not notice when they have been updated.

Three other operations are implemented. These modify the state of amd as a whole, rather than any particular filesystem. The "-l", "-x" and "-D" options have exactly the same effect as amd's corresponding command line options. However, the "-l" option is rejected by amd in version "5.1d" for obvious security reasons.

6 Examples

6.1 Starting Amd

Amd is best started from /etc/rc.local:

```
if [ -f /usr/local/etc/amd.start ]; then
        sh /usr/local/etc/amd.start; (echo -n ' amd')      >/dev/console
fi
```

The shell script, amd.start, contains:

```
PATH=/usr/local/etc:/bin:/usr/bin:/usr/ucb:$PATH export PATH

#
# Either name of logfile or "syslog"
#
LOGFILE=syslog
#LOGFILE=/var/adm/am.log

#
# Figure out whether domain name is in host name
# If the hostname is just the machine name then
# pass in the name of the local domain so that the
# hostnames in the map are domain stripped correctly.
#
case `hostname` in
*.*) dmn= ;;
*) dmn='-d doc.ic.ac.uk'
esac

#
# Zap earlier log file
#
```

This filesystem type is not generally visible externally, but it is possible that the output from `amq -m` may list `inherit` as the filesystem type. This happens when an inherit operation cannot be completed for some reason, usually because a fileserver is down.

5 Run-time Administration

It is sometimes desirable or necessary to exercise external control over some of amd's internal state. To support this requirement, amd implements an RPC interface which is used by the amq program.

Amq provides a variety of operations. With no arguments, amq obtains a brief list of all existing mounts created by amd. This is different from the list displayed by **df**(1) since the latter only includes system mount points. The output from this option includes the following information:

- the automount point,
- the filesystem type,
- the mount map or mount information,
- the internal, or system mount point.

For example:

```
/            auto  "root"                      sky:(pid75)
/homes       auto  /usr/local/etc/amd.homes    /homes
/home        auto  /usr/local/etc/amd.home     /home
/homes/jsp   nfs   charm:/home/charm           /a/charm/home/charm/jsp
/homes/phjk  nfs   toytown:/home/toytown       /a/toytown/home/toytown/ai/phjk
```

The "-m" option displays similar information about mounted filesystems, rather than automount points. The output includes the following information:

- the mount information,
- the mount point,
- the filesystem type,
- the number of references to this filesystem,
- the server hostname,
- the state of the file server,
- any error which has occured.

For example:

```
"root"             truth:(pid602)       auto 1 localhost is up
hesiod.home        /home                auto 1 localhost is up
hesiod.vol         /vol                 auto 1 localhost is up
hesiod.homes       /homes               auto 1 localhost is up
amy:/home/amy      /a/amy/home/amy      nfs  5 amy is up
gould:/home/gould  /a/gould/home/gould  nfs  0 gould is up (Permission denied)
noddy:/home/noddy  /a/noddy/home/noddy  nfs  0 noddy is down
```

will cause `/homes` to be automounted using the *Hesiod* name server with local incremental caching of all succesfully resolved names.

All cached data is forgotten whenever `amd` receives a `SIGHUP` signal. If cache `all` mode was selected, the cache will be reloaded. This can be used to inform `amd` that a map has been updated. In addition, whenever a cache lookup fails and `amd` needs to examine a map, the map's modify time is examined. If the cache is out of date with respect to the map then it is flushed as if a `SIGHUP` had been received.

The `fs` option specifies the name of the mount map to use for the new mount point.[4] The prefix alters the name that is looked up in the mount map. If the prefix is non-null then it is prepended to the name requested by the kernel *before* the map is searched. The prefix can be overridden with the `pref` option.

The server `dylan.doc.ic.ac.uk` has two user disks: `/dev/dsk/2s0` and `/dev/dsk/5s0`. These are accessed as `/home/dylan/dk2` and `/home/dylan/dk5` respectively. Since `/home` is already an automount point, this naming is achieved with the following map entries:

```
dylan        type:=auto;fs:=${map};pref:=${key}/
dylan/dk2    type:=ufs;dev:=/dev/dsk/2s0
dylan/dk5    type:=ufs;dev:=/dev/dsk/5s0
```

4.7 Direct Automount Filesystem (`type:=direct`)

The *direct* filesystem is almost identical to the automount filesystem. Instead of appearing to be a directory of mount points, it appears as a symbolic link to a mounted filesystem. The mount is done at the time the link is accessed.

Direct automount points are created by specifying the `direct` filesystem type on the command line:

```
amd ...  /usr/man auto.direct -type:=direct
```

where `auto.direct` would contain an entry such as:

```
usr/man    -type:=nfs;rfs:=/usr/man \
           rhost:=man-server1  rhost:=man-server2
```

In this example, `man-server1` and `man-server2` are file servers which export copies of the manual pages. Note that the key does *not* have a leading "/".

4.8 Error Filesystem (`type:=error`)

The *error* filesystem type is used internally as a catch-all in the case where none of the other filesystems was selected, or some other error occurred. Lookups always fail with "No such file or directory". All other operations trivially succeed.

The error filesystem is not directly accessible.

4.9 Inheritance Filesystem (`type:=inherit`)

The inheritance filesystem is not directly accessible. Instead, internal mount nodes of this type are automatically generated when `amd` is started with the "-r" option. At this time the system mount table (`/etc/mtab`)[5] is scanned to locate any filesystems which are already mounted. If any reference to these filesystems is made through `amd` then instead of attempting to mount it, `amd` simulates the mount and *inherits* the filesystem. This allows a new version of `amd` to be installed on a live system simply by killing the old daemon and starting the new one.

[4] Arguably this should have been specified with the `${rfs}` option but we are now stuck with it due to historical accident.

[5] in fact many systems maintain this information in the kernel rather than a file.

The exit code from these two programs is interpreted as a UNIX error code. As usual, exit code zero indicates success. To execute the program `amd` splits the string on whitespace to create an array of substrings. Single quotes "'" can be used to quote whitespace if that is required in an argument. There is no way to escape or change the quote character. To run the program `rvdmount` with a host name and filesystem as arguments would be specified by `mount:="/etc/rvdmount rvdmount fserver ${path}"`.

The first element in the array is taken as the pathname of the program to execute. The other members of the array form the argument vector to be passed to the program, *including argument zero*. This means that the split string must have at least two elements. The program is directly executed by `amd`, not via a shell.

If a filesystem type is to be heavily used, it may be worthwhile adding a new filesystem type into `amd`, but for most uses the program filesystem should suffice.

When the program is run, standard input and standard error are inherited from the current values used by `amd`. Standard output is a duplicate of standard error. The value specified with the "-l" command line option has no effect on standard error.

4.5 Symbolic Link Filesystem (`type:=link`)

The *link* filesystem type allows other parts of the filesystem to be referenced as if they were mounted under the automounter. Under SunOS 4.0 this can also be done using the loopback filesystem, however that adds an unnecessary extra mount into the system.

One common use for the symlink filesystem is `/homes` which can be made to contain an entry for each user which points to their (auto-mounted) home directory. Although this may seem rather expensive, it provides a great deal of administrative flexibility.

The value of `fs` option specifies the destination of the link, as modified by the `sublink` option. If `sublink` is non-null, it is appended to `${fs}/` and the resulting string is used as the target.

The `link` filesystem can be though of as identical to the `ufs` filesystem but without actually mounting anything.

An example entry might be

```
jsp   host==charm;type:=link;fs:=/home/charm;sublink:=jsp
```

which would return a symbolic link pointing to `/home/charm/jsp`.

4.6 Automount Filesystem (`type:=auto`)

The *auto* filesystem type creates a new automount point below an existing automount point. Top-level automount points appear as system mount points. An automount mount point can also appear as a sub-directory of an existing mount point. This allows some additional structure to be added, for example to mimic the mount tree of another machine.

The following option may be specified:

`cache` specifies whether the data in this mount-map should be cached. The default value is `none`, in which case no caching is done in order to conserve memory. However, better performance and reliability can be obtained by caching some or all of a mount-map. If the cache option specifies `all`, the entire map is enumerated when the mount point is created. If the cache option specifies `inc`, caching is done incrementally as and when data is required. Some map types do not support cache mode `all`, in which case `inc` is used whenever `all` is requested.

The default cache mode is `none` which means that no data will be cached. Each mount map type has a default cache type, usually `inc`, which can be selected by specifying `mapdefault`.

The cache mode for a mount map can only be selected on the command line. Starting `amd` with the command

```
amd /homes hesiod.homes -cache:=inc
```

```
jsp  host!=charm;type:=nfs;rhost:=charm;rfs:=/home/charm;sublink:=jsp
```

The mount system call and any unmount attempts are always done in a new task to avoid the possibilty of blocking amd.

4.2 Network Host Filesystem (type:=host)

The *host* filesystem allows access to the entire export tree of an NFS server. The implementation is layered above the nfs implementation so keep-alives work in the same way. The only option which needs to specified is rhost which is the name of the fileserver to mount.

The host filesystem type works by querying the mount daemon on the given fileserver to obtain its export list. Amd then obtains filehandles for each of the exported filesystems. Any errors at this stage cause that particular filesystem to be ignored. Finally each filesystem is mounted. Again, errors are logged but ignored. One common reason for mounts to fail is that the mount point does not exist. Although amd attempts to automatically create the mount point, it may be on a remote filesystem to which amd does not have write permission.

Sun's automounter provides a special -hosts map. To achieve the same effect with amd requires two steps. First a mount map must be created as follows

```
/defaults  type:=host;fs:=${autodir}/${rhost}/root;rhost:=${key}
*          opts:=rw,nosuid,grpid
```

and then start amd with the following command

```
amd /n net.map
```

where net.map is the name of map described above. Note that the value of ${fs} is overridden in the map. This is done to avoid a clash between the mount tree and any other filesystem already mounted from the same fileserver.

If different mount options are needed for different hosts then additional entries can be added to the map, for example

```
host2       opts:=ro,nosuid,soft
```

would soft mount host2 read-only.

4.3 UNIX Filesystem (type:=ufs)

The *ufs* filesystem type provides access to the system's standard disk filesystem—usually the Berkeley Fast Filesystem [9]. The following option must be specified:

dev the block special device to be mounted.

A UFS entry might be:

```
jsp   host==charm;type:=ufs;dev:=/dev/xd0g;sublink:=jsp
```

4.4 Program Filesystem (type:=program)

The *program* filesystem type allows a program to be run whenever a mount or unmount is required. This allows easy addition of support for other filesystem types, such as MIT's Remote Virtual Disk (RVD) [10] which has a programmatic interface via the commands rvdmount and rvdunmount.

The following options must be specified:

mount the program which will perform the mount.

unmount the program which will perform the unmount.

which uses a database to map the local hostname into a cluster name. `${cluster}` can then be used as a selector to restrict mounting of replicated data. If this option is not given, `${cluster}` has the same value as `${domain}`. This would be used as follows:

```
amd -C `clustername` ...
```

`-D` *opts*
: controls the verbosity and coverage of the debugging trace; *opts* is a comma separated list of debugging options. The "-D" option is only available if amd was compiled with `-DDEBUG`. The memory debugging facilities are only available if amd was compiled with `-DDEBUG_MEM` (in addition to `-DDEBUG`). The most common options to use are `-D trace` and `-D test` (which turns on all the useful debug options). See the program source for a more detailed explanation of the available options.

Once the command line has been parsed, the automount points are mounted. The mount points are created if they do not already exist, in which case they will be removed when amd exits. Finally, amd disassociates itself from its controlling terminal and forks into the background.

Note: Even if amd has been built with `-DDEBUG` it will still background itself and disassociate itself from the controlling terminal. To use a debugger it is necessary to specify `-D nodaemon` on the command line.

4 Filesystem types

To mount a volume, amd must be told the type of filesystem to be used. Each filesystem type typically requires additional information such as the fileserver name for NFS.

From the point of view of amd, a *filesystem* is anything that can resolve an incoming name lookup. An important feature is support for multiple filesystem types. Some of these filesystems are implemented in the local kernel and some on remote fileservers, whilst the others are implemented internally by amd.

The two common filesystem types are UFS and NFS. Four other user accessible filesystems (`link`, `program`, `auto` and `direct`) are also implemented internally by amd and these are described below. There are two additional filesystem types internal to amd which are not directly accessible to the user (`inherit` and `error`). Their use is described since they may still have an effect visible to the user.

4.1 Network Filesystem (`type:=nfs`)

The *nfs* filesystem type provides access to Sun's NFS. The following options may be specified:

`rhost` the remote fileserver. This must be an entry in the hosts database. IP addresses [8] are not accepted. The default value is taken from the local host name (`${host}`) if no other value is specified.

`rfs` the remote filesystem. If no value is specified for this option, an internal default of `${path}` is used.

NFS mounts require a two stage process. First, the *file handle* of the remote file system must be obtained from the server. Then a mount system call must be done on the local system. Amd keeps a cache of file handles for remote file systems. The cache entries have a lifetime of a few minutes.

If a required file handle is not in the cache, amd sends a request to the remote server to obtain it. Amd *does not* wait for a response; it notes that one of the locations needs retrying, but continues with any remaining locations. When the file handle becomes available, and assuming none of the other locations was successfully mounted, amd will retry the mount. This mechanism allows several NFS filesystems to be mounted in parallel[3]. The first one which responds with a valid file handle will be used.

An NFS entry might be:

[3] The mechanism is general, however NFS is the only filesystem for which the required hooks have been written.

```
amd -p > /var/run/amd.pid ...
```

`-r`
tells `amd` to restart existing mounts (see the Inheritance File System §4.9, p16).

`-t` *afs-timeout.afs-retransmit*
specifies the RPC timeout and retransmit intervals used by the kernel to communicate to `amd`. These are used to set the `timeo` and `retrans` mount options. Amd relies on the kernel RPC retry mechanism to trigger mount retries. The value of this parameter changes the retry interval. Too long an interval gives poor interactive response, too short an interval causes excessive retries.

`-v`
print version information on standard error and then exit. The output is of the form:

```
amd 5.2 of 90/06/23 23:55:04 Rel5.2 #0: Sat Jun 23 16:24:42 PDT 1990
Built by pendry@okeeffe.Berkeley.EDU for a tahoe running bsd44 (big-endian)
Map support for: root, passwd, file, error.
```

The information includes the version number, release date and name of the release. The architecture (§1, p8), operating system (§2, p8) and byte ordering are also printed as they appear in the `${os}`, `${arch}` and `${byte}` variables.

`-w` *wait-timeout*
selects the interval in seconds between unmount attempts after the initial time-to-live has expired. This defaults to 120 seconds (two minutes).

`-x` *opts*
specifies the type and verbosity of log messages. *opts* is a comma separated list selected from the options in table 5. The default logging option, if none is given on the command line, is

Log option	Messages logged
`fatal`	Fatal errors
`error`	Non-fatal errors
`user`	Non-fatal user errors
`warn`	Recoverable errors
`warning`	Alias for `warn`
`info`	Information messages
`map`	Mount map usage
`stats`	Additional statistics
`all`	All of the above

Table 5: Logging options

`-x all,nomap,nostats` which is also reasonable for production use. The `info` messages include details of what is mounted and unmounted and when filesystems have timed out. The messages given by `user` relate to errors in the mount maps, so these are useful when new maps are installed.

The options can be prefixed by the string `no` to indicate that this option should be turned off. For example, to obtain all but `info` messages the option `-x all,noinfo` would be used.

`-y` *yp-domain*
selects an alternate YP domain. This is useful for debugging and cross-domain shared mounting. If this flag is specified, `amd` immediately attempts to bind to a server for this domain.

`-C` *cluster-name*
specifies the name of the cluster of which the local machine is a member. The only effect is to set the variable `${cluster}`. The *cluster-name* is will usually obtained by running another command

point. `type:=auto;cache:=none;fs:=${map}` is the default value for the map options. Default options for a map are read from a special entry in the map whose key is the string `/defaults`. When default options are given they are prepended to any options specified in the mount-map locations as explained in §2.3, p6.

The *options* are any combination of the following:

`-a` *directory*
: specifies the default mount directory. This option changes the variable `${autodir}` which otherwise defaults to `/a`. For example, some sites prefer `/am`.

```
amd -a /am ...
```

`-c` *cache-interval*
: selects the period, in seconds, for which a name is cached by `amd`. If no reference is made to the volume in this period, `amd` discards the volume name to filesystem mapping. Once the last reference to a filesystem has been removed, `amd` attempts to unmount the filesystem. If the unmount fails the interval is extended by a further period as specified by the `-w` command line option or by the `utimeout` mount option. The default *cache-interval* is 300 seconds (five minutes).

`-d` *domain*
: specifies the host's domain. This sets the internal variable `${domain}` and affects the `${hostd}` variable. If this option is not specified and the hostname already contains the local domain then that is used, otherwise the default value of `${domain}` is `unknown.domain`. For example, if the local domain was `doc.ic.ac.uk`, `amd` could be started as follows:

```
amd -d doc.ic.ac.uk ...
```

`-k` *kernel-architecture*
: specifies the kernel architecture of the system. This is usually the output of `arch -k` and its only effect is to set the variable `${karch}`. If this option is not given, `${karch}` has the same value as `${arch}`. This would be used as follows:

```
amd -k `arch -k` ...
```

`-l` *log-option*
: selects the form of logging to be made. Two special *log-option*s are recognised. If *log-option* is the string `syslog`, `amd` will use the **syslog**(3) mechanism. If *log-option* is the string `/dev/stderr`[2], `amd` will use standard error, which is also the default target for log messages. Any other string is taken as a filename to use for logging. Log messages are appended to the file if it already exists, otherwise a new file is created. The file is opened once and then held open, rather than being re-opened for each message. If the `syslog` option is specified but the system does not support syslog or if the named file cannot be opened or created, `amd` will use standard error. Error messages generated before `amd` has finished parsing the command line are printed on standard error. Using `syslog` is usually best, in which case `amd` would be started as follows:

```
amd -l syslog ...
```

`-n`
: normalises the remote hostname before using it. Normalisation is done by replacing the value of `${rhost}` with the primary name returned by a hostname lookup. This option should be used if several names are used to refer to a single host in a mount map.

`-p`
: causes `amd`'s process id to be printed on standard output. This can be redirected to a suitable file for use with kill:

[2] This pathname is interpreted internally by `amd`; a `/dev/fd` driver is not required.

Option	Semantics
`grpid`	Use BSD directory group-id semantics.
`intr`	Allow keyboard interrupts on hard mounts.
`nodevs`	Don't allow local special devices on this filesystem.
`nosuid`	Don't allow set-uid or set-gid executables on this filesystem.
`quota`	Enable quota checking on this mount.
`retrans=`*n*	The number of NFS retransmits made before a user error is generated by a `soft` mounted filesystem, and before a `hard` mounted filesystem reports `NFS server` *yoyo* `not responding still trying`.
`ro`	Mount this filesystem readonly.
`soft`	Give up after *retrans* retransmissions.
`tcp`	Use TCP/IP instead of UDP/IP
`timeo=`*n*	The NFS timeout, in tenth-seconds, before a request is retransmitted.

Table 3: Mount options passed to the mount system call

Option	Semantics
`notimeout`	Configures the mount so that its time-to-live will never expire. This is also the default for some filesystem types.
`ping=`*n*	The interval, in seconds, between keep-alive pings. When four consecutive pings have failed the mount point is marked as hung. This interval defaults to 30 seconds. If the ping interval is negative, no pings are sent and the host is assumed to be always up. Pings are never sent for a TCP mount.
`retry=`*n*	The number of times to retry the mount system call.
`utimeout=`*n*	The interval, in seconds, by which the mount's time-to-live is extended after an unmount attempt has failed. In fact the interval is extended before the unmount is attempted to avoid thrashing. The default value is 120 seconds (two minutes) or as set by the "-w" command line option.

Table 4: Mount options interpreted by amd

`sublink` the subdirectory within the mounted filesystem to which the reference should point. This can be used to prevent duplicate mounts in cases where multiple directories in the same mounted filesystem are used.

`type` the filesystem type to be used. A full description of each type is given in §4, p13.

Superfluous option specifications are ignored and are not reported as errors.

3 Command Line Options

Many of amd's parameters can be set from the command line. The command line is also used to specify automount points and maps.

The general format of a command line is

`amd` *options* directory map-name [`-`*map-options*] ...

For each directory and map-name given, amd establishes an automount point. The *map-options* may be any sequence of options or selectors as described in §2.3, p5. The *map-options* apply only to amd's mount

that when the name from the kernel is expanded prior to a map lookup, these selectors are all defined as empty strings.

`key` the name being resolved. For example, if `/home` is an automount point, then accessing `/home/foo` would set `${key}` to the string `foo`. The key is prefixed by the `pref` option set in the parent mount point. The default prefix is an empty string. If the prefix was `blah/` then `${key}` would be set to `blah/foo`.

`map` the name of the mount map being used.

`path` the full pathname of the name being resolved. For example `/home/foo` in the example above.

Selectors can be negated by using `!=` instead of `==`. For example to select a location on all non-VAX machines the selector `arch!=vax` would be used.

2.3.3 Options

Options are parsed concurrently with selectors. The difference is that when an option is seen the string following the "`:=`" is recorded for later use. As a minimum the `type` option must be specified. Each filesystem type has other options which must also be specified. The filesystem specific options are detailed in §4, p13.

The following options apply to more than one filesystem type:

`delay` the delay, in seconds, before an attempt will be made to mount from the current location. Auxilliary data, such as network address, file handles and so on are computed regardless of this value. A delay can be used to implement the notion of primary and secondary file servers. The secondary servers would have a delay of a few seconds, thus giving the primary servers a chance to respond first.

`fs` the local mount point. The semantics of this option vary between filesystems.

For NFS and UFS filesystems the value of `${fs}` is used as the local mount point. It is important that this string uniquely identifies the filesystem being mounted. To satisfy this requirement, it should contain the name of the host on which the filesystem is resident and the pathname of the filesystem on the local or remote host.

The reason for requiring the hostname is clear if replicated filesystems are considered. If a fileserver goes down and a replacement filesystem is mounted then the *local* mount point *must* be different from that of the filesystem which is hung. Some encoding of the filesystem name is required if more than one filesystem is to be mounted from any given host.

If the hostname is first in the path then all mounts from a particular host will be gathered below a single directory. If that server goes down then the hung mount points are less likely to be accidentally referenced, for example when **getwd**(3) traverses the namespace to find the pathname of the current directory.

The `fs` defaults to `${autodir}/${rhost}${rfs}`. In addition, `rhost` defaults to the local host name (`${host}`) and `rfs` defaults to the value of `${path}`, which is the full path of the requested file; `/home/foo` in the example above (§2.3.2, p9). `${autodir}` defaults to `/a` but may be changed with the "-a" command line option[1]. Note that there is no "/" between the `${rhost}` and `${rfs}` since `${rfs}` begins with a "/".

`opts` the options to pass to the mount system call. A leading "`-`" is silently ignored. The mount options supported generally correspond to those used by **mount**(8) and are listed in table 3. Some additional pseudo-options are interpreted by `amd` and are listed in table 4. Unless specifically overridden, each of the system default mount options applies. Any options not recognised are ignored. If no options list is supplied the string `rw,defaults` is used and all the system default mount options apply.

[1] Sun's automounter defaults to `/tmp_mnt`

Name	Architecture
`alliant`	Alliant FX/4
`arm`	Acorn ARM
`encore`	Encore (reserved)
`fps500`	FPS Model 500
`hp9000`	HP 9000/300 family
`hp9k8`	HP 9000/800 family (reserved)
`ibm032`	IBM RT
`ibm6000`	IBM RISC System/6000
`macII`	Apple Mac II
`mips`	MIPS R2000
`multimax`	Encore Multimax
`orion105`	HLH Orion 1/05
`powernode`	Gould Powernode family
`sun3`	Sun-3 family
`sun4`	Sun-4 family
`tahoe`	Tahoe family
`vax`	DEC Vax

Table 1: Architectures supported by amd

`hostd` is `${host}` and `${domain}` concatenated with a "." inserted between them if required. If `${domain}` is an empty string then `${host}` and `${hostd}` will be identical.

`karch` is provided as a hook for the kernel architecture. This is used on SunOS 4, for example, to distinguish between different `/usr/kvm` volumes. `${karch}` defaults to the value of `${arch}` unless a different value is set with the "-k" command line option.

`os` the operating system. Like the machine architecture, this is automatically determined at compile time. The operating system name can be displayed by running the command `amd -v`. The currently supported operating systems are listed in table 2.

Name	System
`acis43`	4.3 BSD for IBM RT
`aix3`	AIX 3.1
`aux`	System V for Mac-II
`bsd44`	4.4 BSD
`concentrix`	Concentrix 5.0
`fpx4`	Celerity FPX 4.1/2
`hlh42`	HLH OTS 1.*x* (4.2 BSD)
`hpux`	HP-UX 6.*x*
`riscix`	Acorn RISC iX
`sos3`	SunOS 3.4 & 3.5
`sos4`	SunOS 4.*x*
`umax43`	Umax 4.3 BSD
`u2_2`	Ultrix 2.2
`u3_0`	Ultrix 3.0
`utx32`	Gould UTX/32 Rel 2.*x*
`xinu43`	mt Xinu MORE/bsd

Table 2: Operating systems supported by amd

The following selectors are also provided. Unlike the other selectors, they vary for each lookup. Note

`${fs}` was `${autodir}/local/${key}` then after expansion `${fs}` would have the value `/a/local/bin`. Any environment variable can be accessed in a similar way.

Two pathname operators are available when expanding a variable. If the variable name begins with "/" then only the last component of then pathname is substituted. For example, if `${path}` was `/foo/bar` then `${/path}` would be expanded to `bar`. Similarly, if the variable name ends with "/" then all but the last component of the pathname is substituted. In the previous example, `${path/}` would be expanded to `/foo`.

Variable expansion is a two phase process. Before a location is parsed, all references to selectors, *eg* `${path}`, are expanded. The location is then parsed, selections are evaluated and option assignments recorded. If there were no selections or they all succeeded the location is used and the values of the following options are expanded in the order given: `sublink`, `rfs`, `fs`, `opts`, `mount` and `unmount`. Note that expansion of option values is done after *all* assignments have been completed and not in a purely left to right order as is done by the shell. This generally has the desired effect but care must be taken if one of the options references another, in which case the ordering can become significant.

There are two special cases concerning variable expansion. Firstly, before a map is consulted, any selectors in the name received from the kernel are expanded. For example, if the request from the kernel was for `${arch}.bin` and the machine architecture was `vax`, the value given to `${key}` would be `vax.bin`. Secondly, the value of `${rhost}` is expanded and normalized before the other options are expanded. The normalization process strips any local sub-domain components. For example, if `${domain}` was `Berkeley.EDU` and `${rhost}` was initially `snow.Berkeley.EDU`, after the normalization it would simply be `snow`.

2.3.2 Selectors

Selectors are used to control the use of a location. It is possible to share a mount map between many machines in such a way that filesystem location, architecture and operating system differences are hidden from the users. A selector of the form `arch==sun3;os==sos4` would only apply on Sun-3s running SunOS 4.*x*.

Selectors are evaluated left to right. If a selector fails then that location is ignored. Thus the selectors form a conjunction and the locations form a disjunction. If all the locations are ignored or otherwise fail then `amd` uses the *error* filesystem (§4.8, p16). This is equivalent to having a location `type:=error` at the end of each mount-map entry.

The selectors currently implemented are:

`arch` the machine architecture which was automatically determined at compile time. The architecture type can be displayed by running the command `amd -v`. The currently supported architectures are listed in table 1.

`autodir` the default directory under which to mount filesystems. This may be changed by the "-a" command line option. See the `fs` option.

`byte` the machine's byte ordering. This is either `little`, indicating little-endian, or `big`, indicating big-endian. One possible use is to share `rwho` databases (see §6.6, p22). Another is to share ndbm databases, however this use can be considered a courageous juggling act.

`cluster` is provided as a hook for the name of the local cluster. This can be used to decide which servers to use for copies of replicated filesystems. `${cluster}` defaults to the value of `${domain}` unless a different value is set with the "-C" command line option.

`domain` the local domain name as specified by the "-d" command line option. See `host`.

`host` the local hostname as determined by **gethostname**(2). If no domain name was specified on the command line and the hostname contains a period "." then the string before the period is used as the host name, and the string after the period is assigned to `${domain}`. For example, if the hostname is `styx.doc.ic.ac.uk` the `host` would be `styx` and `domain` would be `doc.ic.ac.uk`. `hostd` would be `styx.doc.ic.ac.uk`.

location-list:
    location-selection
    location-list ␣ || ␣ *location-selection*
location-selection:
    location
    location-selection ␣ *location*
location:
    location-info
    -*location-info*
    -
location-info:
    sel-or-opt
    location-info;*sel-or-opt*
    ;
sel-or-opt:
    selection
    opt-ass
selection:
    selector==*value*
    selector!=*value*
opt-ass:
    option:=*value*

Figure 1: Location syntax

A *location-selection* is a list of possible volumes with which to satisfy the request. *location-selection*s are separated by the || operator. The effect of this operator is to prevent use of location-selections to its right if any of the location-selections on its left were selected (see §2.3.2, p7).

The location-selection, and singleton *location-list*, `type:=ufs;dev:=/dev/xd1g` would inform `amd` to mount a UFS filesystem from the block special device `/dev/xd1g`.

The *sel-or-opt* component is either the name of an option required by a specific filesystem, or it is the name of a built-in, predefined selector such as the architecture type. The value may be quoted with double quotes "", for example `type:="ufs";dev:="/dev/xd1g"`. These quotes are stripped when the value is parsed and there is no way to get a double quote into a value field. Double quotes are used to get white space into a value field, which is needed for the program filesystem (§4.4, p14).

A location beginning with a dash "-" is used to specify default values for subsequent locations. Any previously specified defaults in the location-list are discarded. The default string can be empty in which case no defaults apply. The location `-fs:=/mnt;opts:=ro` would set the local mount point to `/mnt` and cause mounts to be read-only by default. Defaults specified this way are appended to, and so override, any global map defaults given with `/defaults`). A `/defaults` value *gdef* and a location list

$$-def_a \sqcup loc_{a_1} \sqcup loc_{a_2} \sqcup -def_b \sqcup loc_{b_1} \ \ldots$$

is equivalent to

$$-gdef;def_a \sqcup loc_{a_1} \sqcup loc_{a_2} \sqcup -gdef;def_b \sqcup loc_{b_1} \ \ldots$$

2.3.1 Variable expansion

To allow generic location specifications `amd` does variable expansion on each location and also on some of the option strings. Any option or selector appearing in the form ${*var*} is replaced by the current value of that option or selector. For example, if the value of `${key}` was `bin`, `${autodir}` was `/a` and

2.1.3 NIS maps

When using NIS (formerly YP), an amd map is implemented directly by the underlying NIS map. Comments and continuation lines are *not* supported in the automounter and must be stripped when constructing the NIS server's database.

NIS maps do not support cache mode `all` and, when caching is enabled, have a default cache mode of `inc` (§4.6, p15).

2.1.4 Hesiod

When the map name begins with the string "`hesiod.`" lookups are made using the *Hesiod* name server. The string following the dot is used as a name qualifier and is prepended with the key being located. The entire string is then resolved in the automount context. For example, if the map name is `hesiod.homes` and the key is `jsp` then *Hesiod* is asked to resolve `jsp.homes.automount`.

Hesiod maps do not support cache mode `all` and, when caching is enabled, have a default cache mode of `inc` (§4.6, p15).

2.1.5 Password

The password map support is unlike the four previous map types. When the map name is the string `/etc/passwd` amd can lookup a user name in the password file and re-arrange the home directory field to produce a usable map entry.

Amd assumes the home directory has the format */anydir/*dom_1*/../*dom_n*/login*. It breaks this string into a map entry where `${rfs}` has the value "*/anydir/*dom_n", `${rhost}` has the value "dom_n *.....* dom_1", and `${sublink}` has the value "*login*".

Thus if the password file entry was

```
/home/achilles/jsp
```

the map entry used by amd would be

```
rfs:=/home/achilles;rhost:=achilles;sublink:=jsp
```

Similarly, if the password file entry was

```
/home/cc/sugar/mjh
```

the map entry used by amd would be

```
rfs:=/home/sugar;rhost:=sugar.cc;sublink:=jsp
```

2.2 Key Lookup

The key is located in the map whose type was determined when the automount point was first created. In general the key is a pathname component. In some circumstances this may be modified by variable expansion (§2.3.1, p6) and prefixing. If the automount point has a prefix, specified by the `pref` option, then that is prepended to the search key before the map is searched.

If the key cannot be found then a *wildcard* match is attempted. Currently this simply attempts to locate a special key "`*`". If the wildcard is not found then an error code is propagated back to the kernel, otherwise amd proceeds as if an exact match had been found. The value field is then used to resolve the mount request (§4, p13).

2.3 Location Format

The value field from the lookup provides the information required to mount a filesystem. The information is parsed according to the syntax shown in figure 1. Note that unquoted whitespace is not allowed in a location description. White space is only allowed, and is mandatory, where shown with the symbol ␣.

obtained. If the the map name begins with the sequence "`hesiod.`" then the *Hesiod* name server will be used. If the map name is the string "`/etc/passwd`" then the password "file" will be used. Otherwise the map name is referenced as a file, then as an ndbm database by that name and then as a YP map. As soon as a valid reference is found the map type is noted for future use but any resources held are released, for example any open file descriptors are closed. The available map types are a configurable component of `amd` and can be displayed by running the command `amd -v`.

By default, `amd` does not cache any data retrieved from a mount map. This is a policy decision based on the observation that most workstations have more available CPU time than spare memory. However, on a multi-user service or larger workstation the faster response provided by a cache can be preferable. The `cache` option can be specified on automount points to alter the caching behaviour (§4.6, p15). Specifying `cache:=mapdefault` selects a suitable default cache mode depending on the map type. The individual defaults are described below.

2.1.1 File maps

When `amd` searches a file for a map entry it does a simple scan of the file and supports both comments and continuation lines.

Continuation lines are indicated by a backslash character ("\") as the last character of a line in the file. The backslash, newline character *and any leading white space on the following line* are discarded. A maximum line length of 2047 characters is enforced after continuation lines are read but before comments are stripped. Each line must end with a newline character; that is newlines are terminators, not separators. The following examples illustrate this:

```
key     valA   valB;   \
          valC
```

specifies *three* locations, and is identical to

```
key     valA   valB;   valC
```

However,

```
key     valA   valB;\
          valC
```

specifies only *two* locations, and is identical to

```
key     valA   valB;valC
```

After a complete line has been read from the file, including continuations, `amd` determines whether there is a comment on the line. A comment begins with a hash ("`#`") character and continues to the end of the line. There is no way to escape or change the comment lead-in character.

Note that continuation lines and comment support *only* apply to file maps and ndbm maps built with the `mk-amd-map` program.

When caching is enabled, file maps have a default cache mode of `all` (§4.6, p15).

2.1.2 ndbm maps

An ndbm map may be used as a fast access form of a file map. A program, `mk-amd-map`, converts a normal map file into an ndbm database. This program supports the same continuation and comment conventions that are provided for file maps. Note that ndbm format files *may not* be sharable across machine architectures. The notion of speed generally only applies to large maps; a small map, less than a single disk block, is almost certainly better implemented as a file map.

ndbm maps do not support cache mode `all` and, when caching is enabled, have a default cache mode of `inc` (§4.6, p15).

1.7 Keep-alives

Use of some filesystem types requires the presence of a server on another machine. If a machine crashes then it is of no concern to processes on that machine that the filesystem is unavailable. However, to processes on a remote host using that machine as a fileserver this event is important. This situation is most widely recognised when an NFS server crashes and the behaviour observed on client machines is that more and more processes hang. In order to provide the possibility of recovery, amd implements a *keep-alive* interval timer for some filesystem types. Currently only NFS makes use of this service.

The basis of the NFS keep-alive implementation is the observation that most sites maintain replicated copies of common system data such as manual pages, most or all programs, system source code and so on. If one of those servers goes down it would be reasonable to mount one of the others as a replacement.

The first part of the process is to keep track of which fileservers are up and which are down. Amd does this by sending RPC requests to the servers' NFS NullProc and checking whether a reply is returned. While the server state is uncertain the requests are re-transmitted at three second intervals and if no reply is received after four attempts the server is marked down. If a reply is received the fileserver is marked up and stays in that state for 30 seconds at which time another NFS ping is sent.

Once a fileserver is marked down, requests continue to be sent every 30 seconds in order to determine when the fileserver comes back up. During this time any reference through amd to the filesystems on that server fail with the error "Operation would block". If a replacement volume is available then it will be mounted, otherwise the error is returned to the user.

Although this action does not protect user files, which are unique on the network, or processes which do not access files via amd or already have open files on the hung filesystem, it can prevent most new processes from hanging.

1.8 Non-blocking Operation

Since there is only one instance of amd for each automount point, and usually only one instance on each machine, it is important that it is always available to service kernel calls. Amd goes to great lengths to ensure that it does not block in a system call. As a last resort amd will fork before it attempts a system call that may block indefinitely, such as mounting an NFS filesystem. Other tasks such as obtaining filehandle information for an NFS filesystem, are done using a purpose built non-blocking RPC library which is integrated with amd's task scheduler.This library is also used to implement NFS keep-alives (§1.7, p3).

Whenever a mount is deferred or backgrounded, amd must wait for it to complete before replying to the kernel. However, this would cause amd to block waiting for a reply to be constructed. Rather than do this, amd simply *drops* the call under the assumption that the kernel RPC mechanism will automatically retry the request.

2 Mount-maps

Amd has no built-in knowledge of machines or filesystems. External *mount-maps* are used to provide the required information. Specifically, amd needs to know when and under what conditions it should mount filesystems.

The map entry corresponding to the requested name contains a list of possible locations from which to resolve the request. Each location specifies filesystem type, information required by that filesystem (for example the block special device in the case of UFS), and some information describing where to mount the filesystem (§2.3.3, p9). A location may also contain *selectors* (§2.3.2, p7).

2.1 Map Types

A mount-map can be implemented as a regular file, an ndbm database a YP map [6] or via the *Hesiod* [7] name server. In addition, the password "file" can be used as a source of map information. A mount-map name is a sequence of characters. When an automount point is created a handle on the mount-map is

is assumed to be functionally identical to the target filesystem. By default there is a one-to-one mapping between the pair (host, filesystem) and the local mount point so this assumption is valid.

1.4 Operational Principles

Amd operates by introducing new mount points into the namespace. The kernel sees these mount points as NFS [3] filesystems being served by amd. Having attached itself to the namespace, amd is now able to control the view the rest of the system has of those mount points. RPC [4] calls are received from the kernel one at a time.

When a *lookup* call is received amd checks whether the name is already known. If it is not, the required volume is mounted. A symbolic link pointing to the volume root is then returned. Once the symbolic link is returned, the kernel will send all other requests direct to the mounted filesystem.

If a volume is not yet mounted, amd consults a configuration *mount-map* corresponding to the automount point. Amd then makes a runtime decision on what and where to mount a filesystem based on the information obtained from the map.

Amd does not implement all the NFS requests; only those relevant to name binding such as *lookup*, *readlink* and *readdir*. Some other calls are also implemented but most simply return an error code; for example *mkdir* always returns "Read-only filesystem".

1.5 Mounting a Volume

Each automount point has a mount map. The mount map contains a list of key–value pairs. The key is the name of the volume to be mounted. The value is a list of locations describing where the filesystem is stored in the network. In the source for the map the value would look like

$$location_1 \quad location_2 \quad \ldots \quad location_n$$

Amd examines each location in turn. Each location may contain *selectors* which control whether amd can use that location. For example, the location may be restricted to use by certain hosts. Those locations which cannot be used are ignored.

Amd attempts to mount the filesystem described by each remaining location until a mount succeeds or amd can no longer proceed. The latter can occur in three ways:

- If none of the locations could be used, or if all of the locations caused an error, then the last error is returned.
- If a location could be used but was being mounted in the background then amd marks that mount as being "in progress" and continues with the next request; no reply is sent to the kernel.
- Lastly, one or more of the mounts may have been *deferred*. A mount is deferred if extra information is required before the mount can proceed. When the information becomes available the mount will take place, but in the mean time no reply is sent to the kernel. If the mount is deferred, amd continues to try any remaining locations.

1.6 Automatic Unmounting

To avoid an ever increasing number of filesystem mounts, amd removes volume mappings which have not been used recently. A time-to-live interval is associated with each mapping and when that expires the mapping is removed. When the last reference to a filesystem is removed, that filesystem is unmounted. If the unmount fails, for example the filesystem is still busy, the mapping is re-instated and its time-to-live interval is extended. The global default for this grace period is controlled by the "-w" command-line option (§3, p12). It is also possible to set this value on a per-mount basis (§2.3.3, p10).

1 Overview

Amd maintains a cache of mounted filesystems. Filesystems are *demand-mounted* when they are first referenced, and unmounted after a period of inactivity. Amd may be used as a replacement for Sun's **automount**(8) [1, 2] program. It contains no proprietary source code and has been ported to numerous flavours of UNIX (see table 2, p8).

Amd was designed as the basis for experimenting with filesystem layout and management. Although amd has many direct applications it is loaded with additional features which have little practical use. At some point the infrequently used components may be removed to streamline the production system.

The fundamental concept behind amd is the ability to separate the name used to refer to a file from the name used to refer to its physical storage location. This allows the same files to be accessed with the same name regardless of where in the network the name is used. This is very different from placing `/n/hostname` in front of the pathname since that includes location dependent information which may change if files are moved to another machine. By placing the required mappings in a centrally administered database, filesystems can be re-organised without requiring changes to password files, shell scripts and so on.

1.1 Filesystems and Volumes

Amd views the world as a set of fileservers, each containg one or more filesystems where each filesystem contains one or more *volumes*. Here the term volume is used to refer to a coherent set of files such as a user's home directory or a TEX distribution.

In order to access the contents of a volume, amd must be told in which filesystem the volume resides and which host owns the filesystem. By default the host is assumed to be local and the volume is assumed to be the entire filesystem. If a filesystem contains more than one volume, then a *sublink* is used to refer to the sub-directory within the filesystem where the volume can be found.

1.2 Volume Naming

Volume names are assumed to be unique across the entire network. A volume name is the pathname to the volume's root as known by the users of that volume. Since this name uniquely identifies the volume contents, all volumes can be named and accessed from each host, subject to administrative controls.

Volumes may be replicated or duplicated. Replicated volumes contain identical copies of the same data and reside at two or more locations in the network. Each of the replicated volumes can be used interchangeably. Duplicated volumes each have the same name but contain different, though functionally identical, data. For example, `/vol/tex` might be the name of a TEX distribution which varied for each machine architecture.

Amd provides facilities to take advantage of both replicated and duplicated volumes. Configuration options allow a single set of configuration data to be shared across an entire network by taking advantage of replicated and duplicated volumes.

Amd can take advantage of replacement volumes by mounting them as required should an active fileserver become unavailable.

1.3 Volume Binding

UNIX implements a namespace of hierarchically mounted filesystems. Two forms of binding between names and files are provided. A *hard link* completes the binding when the name is added to the filesystem. A *soft link* delays the binding until the name is accessed. An *automounter* adds a further form in which the binding of name to filesystem is delayed until the name is accessed.

The target volume, in its general form, is a tuple (host, filesystem, sublink) which can be used to name the physical location of any volume in the network.

When a target is referenced, amd ignores the sublink element and determines whether the required filesystem is already mounted. This is done by computing the local mount point for the filesystem and checking for an existing filesystem mounted at the same place. If such a filesystem already exists then it

This document describes release "5.1d" of amd (also known as 5.1.1.7).

Amd – An Automounter

Jan-Simon Pendry
<jsp@doc.ic.ac.uk>
Department of Computing,
Imperial College,
London SW7 2BZ

May 1990
Printed June 23, 1990

Abstract

amd, *éiémdí:*, a program providing an automounting service [fr. *A*uto*M*ount *D*aemon]
automount, *ö'tō-mownt, v.i.* to mount a filesystem automatically [Gr. *automatos*, self-moving + UNIX **mount**(2)]

An *automounter* maintains a cache of mounted filesystems. Filesystems are mounted on demand when they are first referenced, and unmounted after a period of inactivity.

Amd may be used as a replacement for Sun's automounter. The choice of which filesystem to mount can be controlled dynamically with *selectors*. Selectors allow decisions of the form "hostname is *this*" or "architecture is not *that*." Selectors may be combined arbitrarily. Amd also supports a variety of filesystem types, including NFS, UFS and the novel *program* filesystem. The combination of selectors and multiple filesystem types allows identical configuration files to be used on all machines so reducing the administrative overhead.

Amd ensures that it will not hang if a remote server goes down. Moreover, amd can determine when a remote server has become inaccessible and then mount replacement filesystems as and when they become available.

Amd contains no proprietary source code and has been ported to numerous flavours of UNIX.