The Automounter

Brent Callaghan

Tom Lyon

Sun Microsystems, Inc.
2550 Garcia Avenue.
Mountain View, Ca. 94043

ABSTRACT

This paper describes the automounter – an automatic filesystem mounting service distributed with Sun Microsystems version of the Unix® operating system (SunOs). The automounter detects access to remote filesystems and mounts them on demand. This action is transparent to users and programs. Automounted filesystems are automatically unmounted after a period of inactivity. The map files that control the automounter can specify multiple locations for filesystems replicated across a network and can describe mount hierarchies. Automount maps can be administered on a single machine through local files or across a Yellow Pages domain.

1. Introduction

The automounter was originally developed as a component of Sun's Network Software Environment (NSE™) [1]. An important requirement was for seamless access to files whether they be on a local disk or on a remote server. A file anywhere on the network could be accessed with a Unix pathname, leaving the automounter the task of locating the file and mounting the filesystem containing it. The utility of the automounter was not confined to the NSE – it was quickly found to be a useful service in a general Unix environment, particularly in a large network where it was neither practical nor desirable to mount every exported filesystem from every server. The automounter is now a service available in SunOs version 4.0. As a user-level server for Sun Microsystem's Network File System (NFS™) [2], the automounter does not require any explicit support from the Unix kernel. It can be compiled and run on any version of Unix that supports the NFS protocol.

2. The Automount Server

The automount server is a daemon that provides NFS service at one or more mount points in the filesystem. The Unix kernel uses remote procedure calls to communicate with the daemon, just as it would for a remote NFS server. At any of its mount points, the daemon can intercept a request to access a remote filesystem, mount it if it's not already mounted, and return a symbolic link to the mount point.

As a simple example, consider */src* as an automounter mount point that provides access to source trees. Upon a reference to */src/bsd* the kernel would send an NFS *lookup* request to the automounter for the pathname component *bsd*. The automounter would lookup *src* in a source tree map, find the location of the corresponding source tree *server:/directory*, create a mount point in its */tmp_mnt* directory, and mount the source tree. To the original *lookup* request in */src*, the response would be a symbolic link to the mount point in */tmp_mnt*. The program that made the reference to */src/bsd* would have no way of knowing that the symbolic link did not exist prior to the *lookup* and that the source tree was not previously mounted.

2.1. Automount Behavior

The automounter needs to support only a small subset of the NFS protocol. The required subset depends on whether the automounter is emulating a symbolic link or a directory of symbolic links (Figure 1 overleaf). In particular, the automounter does not need to support *read* or *write* requests. It exists only to provide a name binding and mounting service. An Automounted file system is mounted within the */tmp_mnt* directory. The automounter merely creates a symbolic link to a mount point within */tmp_mnt* and hands it back to the kernel in response to a *readlink* request. From then on the kernel accesses the file system through the actual server.

At startup the automounter opens a UDP socket and registers it with the *portmapper* service as an NFS server port. It then forks off a server daemon that listens for NFS requests on the socket. The parent process proceeds to mount the daemon at its mount points within the filesystem. Through the *mount* system call it passes the server daemon's socket address and an NFS *filehandle* that is unique to each mount point. The daemon uses the filehandle to identify the mount point that is the source of subsequent requests from the client (kernel). Once the parent program has completed its mounts, it exits. The server daemon serves its mount points using one of two emulations at each mount point: symbolic link and directory of symbolic links.

2.2. Emulations

Symbolic Link

This emulation uses an entry in a *direct map*. In a direct map each entry has a pathname for an automount mount point, a remote filesystem location and mount options that correspond to this mount point. The automounter responds as if there is a symbolic link at its mount point. In response to a *getattr* request the automounter describes itself as a *symbolic link*. When the kernel follows with a *readlink* the automounter returns a path to the *real* mount point for the remote filesystem in */tmp_mnt.*

Directory of Symbolic Links

This emulation uses an *indirect* map. In response to a *getattr* request, the automounter describes itself as a *directory*. When given a *lookup* request it takes the name to be looked up and searches the indirect map. If it finds the name in the map, it returns the attributes of a symbolic link. In response to a *readlink* it returns a path to the mount point in */tmp_mnt* for this directory entry. A *readdir* of the automounter's mount point returns a list of entries that are currently mounted.

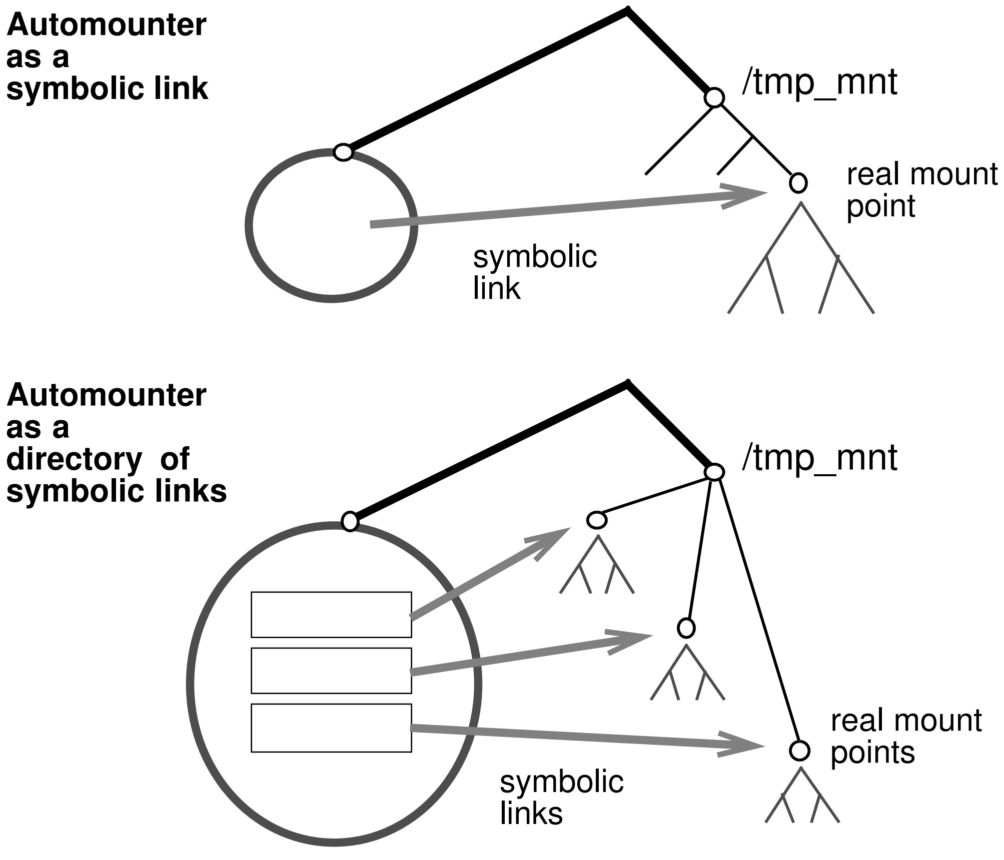

Figure 1

Whatever the emulation, the automounter's responses to *lookup*, *getattr*, and *readlink* are the same whether the filesystem requested is mounted or not. If the filesystem is already mounted and the symbolic link has been read recently, the cached symbolic link is returned immediately. Since the automounter is on the same host, the response is much faster than a *readlink* to a remote NFS server. If the filesystem is not mounted then there is a small delay while it is mounted on a mount point in */tmp_mnt.*

The automounter maintains up-to-date knowledge of *all* the NFS mounts on its host through an internally cached version of the system mount table */etc/mtab*. The mount table contains not only entries for mounts done by the automounter itself, but also any mounts from the *mount* command. Prior to doing a mount, the automounter checks the mount table to see if the same remote filesystem has already been mounted with the same mount options. If so, it avoids duplicating the mount and immediately returns a symbolic link to the existing mount point. Whenever the cached mount table is consulted, the modification time of the */etc/mtab* file is checked. If it indicates that the */etc/mtab* file has been updated, the cache is updated. The modification time checking is done prior to any consultation of the cached mount table. A SIGHUP signal received by the daemon will also provoke a check of the cache. This may be necessary to make the automounter aware of a change in the mount table caused by a *umount* of an automounted file system by the system administrator.

The automounter checks whether a filesystem location is actually on its own host system. Since no mount is required to access a local filesystem, a symbolic link to the requested directory is returned.

2.3. Automatic Unmounting

To avoid a gradual buildup of remote mounts, the automounter unmounts filesystems that appear not to have been used for some time. This time period is 5 minutes by default unless it is changed by a command line option. For each *symbolic link* that the automounter creates to reference a mounted filesystem, it maintains a timestamp. The timestamp is updated whenever the symbolic link is used. Periodically, the automounter checks the timestamps on its symbolic links. If the timestamp indicates that the symbolic link has not been used for some time, it invalidates it. This forces a map lookup the next time the symbolic link is referenced.

A timestamp is also maintained for each *remote filesystem* that the automounter has mounted. The timestamp is updated whenever a symbolic link is used that references the filesystem mount point. The automounter allows many symbolic links to refer to a common mount point. If a filesystem timestamp indicates that it has not been referenced for some time, the automounter attempts to unmount it. The unmount will fail if the filesystem is in-use, for example, a process may be using a subdirectory as its current directory. In this case, the automounter just updates the filesystem timestamp.

3. The Automount Map

There is a map associated with every automounter mount point. A map is a table of entries that describes the locations of filesystems. It is consulted whenever the location and mount options of a remote filesystem are required. The information in the map is not cached, any update to a map will be used when the map is next consulted.

The connection is made between each mount point and map at startup of the automounter. Mount points and map names can be given in a Yellow Pages [3] map *auto.master* (see later), a local file named by the *–f* command line option, and/or command line arguments. The following command line starts an automounter that serves three mount points:

```
automount /net -hosts   /home auto.home -intr,nosuid   /src
/etc/srcmap
```

The map for `/net` is a built-in map (see later), `/home` will use a YP map and has default mount options, `/src` will use a local file as a map. If the mount points (`/net`, `/home`, and `/src`) don't already exist, the automounter will create them as well as the mount directory */tmp_mnt* . The map name is assumed initially to refer to a local file. If the file doesn't exist, then it is assumed to be the name of a Yellow Pages map. As the command line example above demonstrates, a single automount daemon can serve many mount points.

3.1. A Direct Map

The entries in a *direct* map associate a *pathname* with a filesystem location. The pathname names a mount point where the automount daemon emulates a *symbolic link*. At startup, the automounter mounts its daemon at the pathname given as the key for every entry in the direct map. As a result, an entry added to a *direct* map will have no effect after the automounter has started. The direct map as a whole is not associated with any mount point – it is associated with a *dummy* mount point "/–" to please the command line parser. The following example is a direct map:

```
/usr/man          -ro,intr    servant:/usr/manpages/3.2
/usr/local        -ro,intr    devnull:/usr/local/sun3
/usr/src          -rw,intr    source:/srcs/3.2
```

3.2. An Indirect Map

The entries in an *indirect* map associate a *simple name* with a filesystem location. The map is associated with a mount point where the automount daemon emulates a *directory* of symbolic links. The entries in the map correspond to entries in the emulated directory. The behavior of this directory is unusual in that a *readdir* request will reveal only the currently active symbolic links – it will not enumerate the map. This prevents shell filename expansion and commands like *find* from inadvertently forcing the automounter to mount every entry in the map.

```
man               -ro,intr    servant:/usr/manpages/3.2
local             -ro,intr    devnull:/usr/local/sun3
src               -rw,intr    source:/srcs/3.2
```

The *key* to a map entry is a *pathname* for an direct map and a *simple name* for an indirect map. The rest of the map entry shares the same syntax.

3.3. Map Syntax

An automount map contains one or more entries. An entry may be either a line in a file or an entry in a Yellow Pages map. Within a file, a newline protected with a backslash may be used to continue long entries onto successive lines. A hash symbol "#" comments out the rest of an entry. Tokens within an entry are any sequence of characters delimited by whitespace (spaces, tabs or escaped newlines).

3.4. A Simple Map Entry

A simple map entry contains a key, optional mount options, and a location. The format of the location is a server hostname followed by a colon and a pathname for the exported file system on the server. The mount options need to be supplied only if they are different from those given as a default for the whole map. The following example shows an indirect map with three entries.

```
# Key             Mount opts  Location
asparagus                     veg:/usr/greens
parsley           -ro,intr    veg:/usr/greens/cond
carrot                        tuber:/var/crunchy
```

3.5. Subdirectory Specification

In the indirect map example, the map entries for `asparagus` and `parsley` refer to the same remote file system `veg:/usr`. As the map is shown, the automounter would have to perform a separate mount for each entry. These two separate mounts can be reduced to a single mount at a higher level in the filesystem, where the separate map entries refer to subdirectories within this common filesystem. This can be achieved by replacing a slash by a colon after the common part of the pathname. The following example demonstrates this:

```
asparagus                        veg:/usr:greens
parsley            -ro,intr      veg:/usr:greens/cond
carrot                           tuber:/var/crunchy
```

The automounter appends the pathname after the second colon to the symbolic link to the mount point in */tmp_mnt*. Other map entries that reference subdirectories of this mount are saved the overhead of doing a mount if the filesystem is mounted already.

3.6. Locations and Replication

The automounter provides a unique mount service with its provision for multiple locations for a read-only filesystem that is replicated across the network. A good example is the *man* pages. In a large network, the current set of manual pages may be available from a number of servers. It doesn't matter which server they are mounted from, just as long as the server is up and running. Multiple mount locations can be expressed to the automounter simply as a list of mount locations in a map entry e.g.

```
/usr/man           doh:/usr/man  re:/usr/man  me:/usr/man
```

Here the man pages can be mounted from either of the servers *doh*, *ray* or *me*. From this list of servers the automounter first selects those that are "local" (on the local net) and "pings" these servers. There is no ordering implied by the list. To ping the servers, it launches a series of RPC requests to the null procedure of the mount service on each server and offers the mount to the first server to respond. If the original list of servers contained non-local servers and no response is received from any of the local servers, it repeats the ping to all the servers in the list.

The redundancy that is made available by multiple locations is enjoyed only at mount time. There is no status checking of the mounted-from server by the automounter once the mount has been done. If the server goes down while the mount is in effect, the filesystem becomes unavailable. In this situation a user could wait 5 minutes until the auto-unmount takes place and try again; next time around the automounter will choose one of the available servers from the list.

3.7. Hierarchical Mounts

When filesystems are mounted hierarchically, each filesystem is mounted on a subdirectory within another filesystem. The automount map syntax allows a hierarchy of mounts to be described. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy. The individual mounts within the hierarchy must all be NFS mounts, but they can be from different locations and have their own mount options.

An important concept here is the *mount root*. The symbolic link returned by the automounter to the kernel's *lookup* request is a path to the mount root. This is the root of the hierarchy that is mounted within the */tmp_mnt* directory. The location in a map entry is preceded by a path that specifies the mount point relative to the mount root. In the trivial case of a single mount the mount point may be omitted – it is assumed to be *at* the mount root or "/". The simple map entries in the preceding examples had implied mount roots of "/". e.g.

```
parsley            -ro,intr      veg:/usr:greens/cond
```

is the same as

```
parsley   /        -ro, intr     veg:/usr:greens/cond
```

The mount point specification becomes more important when mounting a hierarchy. The automounter must have a mount point for each mount within the hierarchy. This is best explained using an example:

```
/usr/local \
      /            -ro,intr      devnull:/export/local \
      /share       -ro,intr      devnull:/export/local/share \
      /src         -ro,intr      devnull:/export/local/src
```

There is a single entry here for the key */usr/local*. Readability has been improved by splitting the entry into four lines and indenting the continuation lines. The mount points used here for the hierarchy are `/`, `/share` and `/src`. Note that these mount point paths are relative to the *mount* root – not the host's *filesystem* root. The first entry in this example has "/" as its mount point. It is mounted *at* the mount root. There is no requirement that the first mount of a hierarchy be at the mount root. The automounter will happily issue *mkdir*'s to build a path to the first mount point if it isn't at the mount root.

3.8. Substitutions

String substitutions can be made within a map entry at runtime. The value of an environment variable can be substituted by prefixing its name with a dollar sign. Braces can be used to delimit the name of the environment variable from appended letters or digits. Environment variables may be inherited from the environment or placed explicitly into automounter's environment with the "–D" command line option. If the environment variable "ARCH" is not defined, the automounter sets it to the output of the *arch* command. This is useful in the case where a common Yellow Pages map is used by clients of different architectures. Architecture-dependent executables can be mounted from different directories e.g.

```
exec                mutt:/export/exec/$ARCH  jeff:/export/exec/$ARCH
```

An ampersand character can be used to substitute the value of the key. This is most useful in the case of a *catch-all* key. A map entry that has an asterisk as its key will catch any key not matched by any previous entry. The catch-all entry may be used to replace a much larger map that would have contained a repeated pattern:

```
peanuts             peanuts:/home/peanuts
snoopy              snoopy:/home/snoopy
linus               linus:/home/linus
  :
```

could be replaced by

```
*                   &:/home/&
```

3.9. Built-in Maps

The automounter provides two built-in maps. The *–hosts* map gives access to the exported NFS filesystems of any server in the network. With the *–hosts* map mounted on */net*, a "`cd /net/flower`" would change the current directory to the root directory of `flower`. Given a server's hostname, the automounter contacts its mount service and requests a list of the exported filesystems. From this export list a hierarchical map entry is built internally and this is used to mount the exported hierarchy from the server. Given the following export list from server *flower*:

```
/usr/man
/usr/src
/usr/src/sccs
```

The automounter would build the following internal map entry:

```
flower \
   /usr/man       -rw         flower:/usr/man \
   /usr/src       -rw         flower:/usr/src \
   /usr/src/sccs  -rw         flower:/usr/src/sccs
```

and use this map entry to mount all the mountable filesystems exported from *flower*. Since the root filesystem has not been exported, the mount root for *flower* will contain only the */usr* mount point created by the automounter. Filesystems that do appear in the export list may not be mountable because the automounter's host machine doesn't have permission.

As an NFS server, the automounter sees only one component of a path to a remote filesystem through its mount point. Since it has no way of knowing in advance which exported filesystem will be accessed on the server, it must mount *all* the server's exported file systems. This may dismay casual users who notice that a reference to single filesystem through the *–hosts* map also mounts many unwanted filesystems. In practice this is not too much of a problem. A dozen mounts can be done in just a few seconds. The *–hosts* map is particularly useful for casual browsing of servers around a network. A specialized map should be created if frequent access to a specific file system is required.

The *–passwd* map was included to provide easy access to user's home directories around the network. It works only for a specific format of home directory path that includes the hostname of the server that exports the home directory. At Sun we use the format "/home/*server*/*loginname*". Given the login name of a user, the automounter makes a call to *getpwnam()* to get the user's home directory path. It uses the server name in the middle component of the path to build an internal map entry:

```
loginname          -ro,nosuid  server:/home/server:loginname
```

As an additional feature, given a *lookup* of the tilde "~" character, the user's *uid* is extracted from the credentials passed with the NFS request and the home directory path is obtained with a call to *getpwuid()*. A symbolic link containing a reference to a mount point for the *–passwd* map and a tilde as the next component would point back into the user's own home directory. This could be used as part of a scheme to move mail files from */usr/spool/mail* into users' home directories.

4. Administration

The automounter, its mount points, and maps can be administered on a single host through the use of regular files or across a whole Yellow Pages domain with YP maps. Every client that hosts an automounter must have an entry in its */etc/rc.local* file to start the automounter at boot time.

4.1. Local Administration

We have used the convention of putting maps that are files in */etc* and use a prefix of "auto." on the map name. In the absence of the Yellow Pages the *–hosts* and *–passwd* built-in maps would continue to work but only for entries in the */etc/hosts* and */etc/passwd* files.

A file map entry can be a reference to a YP map. If in the course of scanning a local map for key, the automounter finds a key with a prepended plus "+" sign, it treats the key as the name of a YP map to be consulted. This feature may be used to interpose map entries that are specific to the host before the YP map is consulted.

```
bsd4.2             berk:/usr/src:bsd4.2
bsd4.3             berk:/usr/src:bsd4.3
+auto.src
*                  &:/usr/src
```

The map above provides access to various source trees. If the key is not `bsd4.2` or `bsd4.3` the YP map `auto.src` is consulted for the key. If it's not found there either it is caught by the "catchall" entry which assumes that the key is a server name and attempts to mount `/usr/src` from that server.

4.2. Yellow Pages Administration

Given a map name, the automounter first checks for a local file. If it's not a file it assumes that the name refers to a Yellow Pages map. The Yellow Pages service is not *required* by the automounter, but when it's available it can be used to administer the mount points for clients across an entire YP domain. Changes in the locations of file systems in a network need to be reflected only in the appropriate YP maps. The changes will be effected transparently on the clients who need not be aware of changes in mount location, mount options or the addition of new map entries.

A local map file can be converted to a Yellow Pages map by the YP administrator. A "catchall" entry will continue to work in a YP map. If a lookup for a key in a YP map fails, the automounter tries again using a "*" key and uses the catchall entry if one exists in the map.

4.3. Auto.master

At startup, the automounter checks for the presence of a YP map called *auto.master*. The syntax here is not that of the direct or indirect maps. Each entry contains a mount point, a map name, and optional default mount options for the map.

```
# Mount point      Map          Mount options
/-                 auto.direct -ro,intr
/home              auto.home   -rw,intr,secure
/net               -hosts
```

Changes made to this map by the YP administrator will affect every client in the YP domain that uses the automounter. Since *auto.master* is read only when the automounter is started up, any changes made to the map will not be effective until the automounter is restarted. A client is not forced to use the *auto.master* map either in whole or in part. Additional mount points and maps can be specified by an individual client either on the command line or in a file. A client can cancel a mount point in *auto.master* with a *–null* map on the command line. The entire auto.master can be disregarded by setting the *–m* flag on the command line.

A system administrator can exercise a great degree of control over the NFS mounts of each client. File systems can be added and moved about the network without the knowledge of the clients. Only the YP maps that control their automounters need to be updated.

5. Future Work

The current implementation requires a response to be sent in reply to an NFS request before the next request can be serviced. For most RPC services this doesn't present too much of a performance problem if requests can be serviced quickly. The automounter can respond quickly to NFS requests that reference cached symbolic links but it can be subject to substantial delays if a request requires a mount from a server that is slow or is not responding. Not only will the current request be delayed, but *all* requests will be delayed until the automounter responds. This unevenness of response time could become intolerable on a large multi-user machine with a single automounter daemon. The problem could be somewhat alleviated by forking off a separate daemon to serve each mount point at the cost of increasing memory usage. A solution we would like to pursue is to make the automount daemon support multiple threads of execution according to a Lightweight Process model[4]. This would allow the automounter to handle a number of NFS requests concurrently.

The locations in an automount map currently use hard-coded server and exported filesystem names. We would like to allow a filesystem location to be given as a name that could be mapped to a location (server:/dir) by a name binding service. This facility would allow individual servers in a network to "advertise" their exported filesystems.

The *-hosts* map permits easy access to all the *exported* filesystems from a server. This is commonly misunderstood to mean "access to the *mounted* filesystems of any host". This map cannot be used to browse the mounted filesystems of a diskless machine. We would like to extend the mount protocol to allow the automounter to find out what filesystems a host has mounted and allow it to reproduce those mounts. This facility would greatly improve the network transparency offered by the *-hosts* map.

The mount points served by the automounter are fixed at startup. The only way to add new mount points is to terminate the automount daemon with a *kill* command and start it up again. We would prefer to be able to offer uninterrupted automount service while such a change is made. An alternative structure for the automounter, would be to split it into two commands: a command that simply forks the server daemon with no mount points assigned, and a process invoked through the *mount* command that could mount the daemon at any given mount point and assign a map.

6. Acknowledgments

The development of the automounter has been spurred and encouraged by many people within Sun. Brad Taylor wrote the user-level NFS server skeleton on which the automounter was based. Bob Gilligan, Bill Shannon, Bob Lyon, Daniel Steinberg, Carl Smith, John Pope, David DiGiacomo, Marty Hess, and Dave Brownell all suffered early versions, described the bugs and provided many useful suggestions.

REFERENCES

[1] "Introduction to the NSE", Sun Microsystems, Inc. (1988)

[2] R. Sandberg *et al*, "Design and Implementation of the Sun Network Filesystem", *USENIX Conference Proceedings*, Portland, Summer, 1985.

[3] P. Weiss, "Yellow Pages Protocol Specification", Sun Microsystems, Inc. Technical Report, 1985.

[4] J. H. Kepecs, "Lightweight Processes for UNIX Implementation and Applications", *USENIX Conference Proceedings*, Portland, Summer, 1985

[5] David Hendricks, "The Translucent File Service", *EUUG Conference*, Portugal, 1988.