The Automounter

Conference Paper · January 1989

Source: CiteSeer

CITATIONS
10

READS
24

2 authors, including:

Tom Lyon
Princeton University
18 PUBLICATIONS **721** CITATIONS

SEE PROFILE

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.

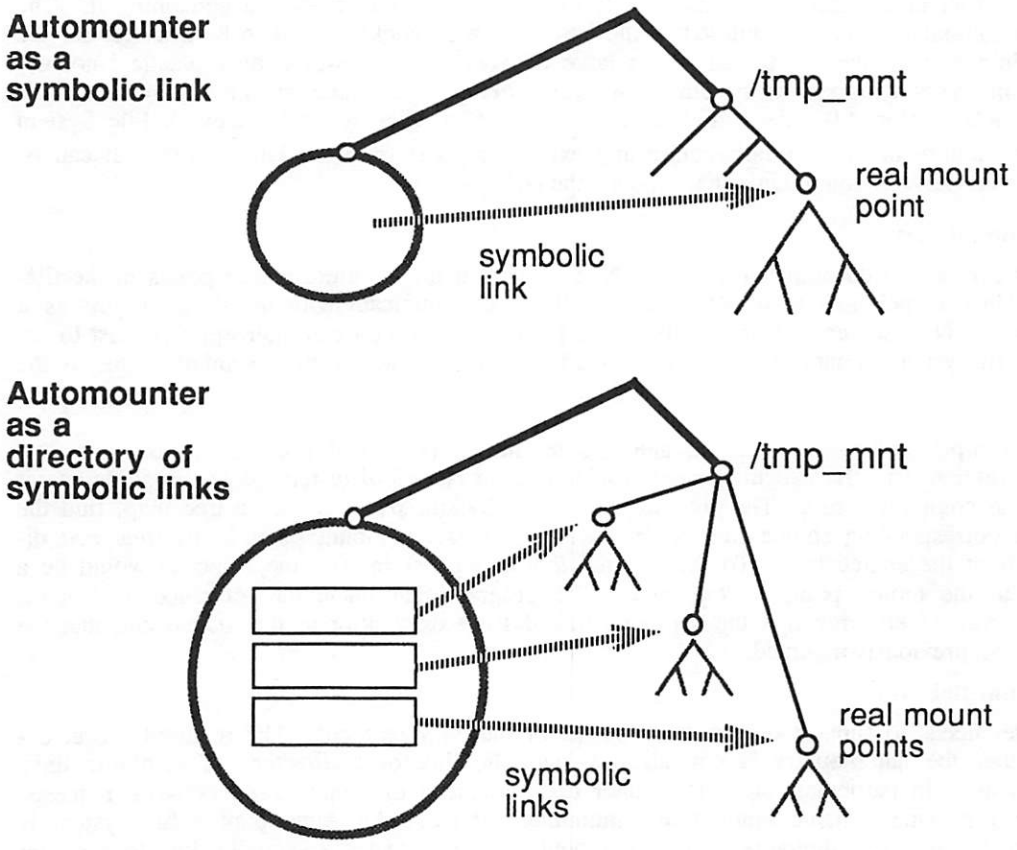

Figure 1

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.