NEXTSTEP In Focus, Summer 1993 (Volume 3, Issue 3).

The NetInfo Lookup ServerÐlookupd

Alan M. Marcum

Relatively few processes communicate directly with NetInfo. Yet, many processes need information that resides in NetInfo. How do these processes get that information? They use an intermediary: **lookupd**.

This article explains what **lookupd** is and what it does, and gives you some tips on spotting and fixing problems that involve it.

WHAT IS LOOKUPD?

lookupd is a daemon that simplifies the tasks of the UNIX library routines that need system and network administration information. These routines, such as getpwuid(), gethostbyname(), and getgrent(), are principally part of the C library (also known as libc). They access information like user names, computer addresses, and group IDs.

lookupd gets information from NetInfo, the DNS, NIS, and the UNIX system files. (The UNIX system files are actually accessed as part of consulting NIS.) It uses Mach messages and SunRPCs to provide information to callers, usually libc routines.

NeXT created **lookupd** to avoid rewriting all the libc routines any time a new information service is added. If, for example, we wanted client processes to get network administration information from a relational database, we could modify **lookupd** so it referenced the information in the database. This would be easier than modifying all the appropriate libc routines to give them relational database access.

In addition to providing a centralized data access service, **lookupd** also caches some information. This caching improves system responsiveness and decreases

network traffic.

Figure 1 shows how **lookupd** is used. Most client processes that need information get it through the libc routines. Those routines call **lookupd**, which gets information from NetInfo, the DNS, NIS, and (through NIS) the UNIX system files. Note, though, that **lookupd** doesn't prevent some client processes from accessing the information sources directly. Furthermore, even though most access to **lookupd** is through libc, clients can also invoke **lookupd**'s services directly.

Fig1_Lookup_4Sources.eps ¬

Figure 1: *Clients get NetInfo, DNS, and NIS information directly or through libc and lookupd.*

In this figure, the first client process could be one like **sendmail**. **sendmail** calls libc to resolve host names, NetInfo to get configuration information, and the DNS to look up MX records. It also consults NIS in response to NIS map access directives in its configuration file.

The last process in the figure might be one like **loginwindow**, which uses libc to do things like look up user names. **loginwindow** also communicates directly with **lookupd** for cache management.

When searching for information, **lookupd** consults its information sources in the following order:

1. **lookupd**'s cache
2. The NetInfo domain hierarchy
3. The DNS, if appropriate
4. NIS

This search order applies in all of NeXT's releases through NEXTSTEP 3.1. To find out about the DNS see Albitz and Liu 1992, and Nemeth, Snyder, and Seebass 1989; to find out about NIS see Nemeth, Snyder, and Seebass 1989, and Stern 1991.

LOOKUPD CACHING

There are various types of **lookupd** caches, and each type is flushed and refreshed differently. The following sections provide details.

Password entries cache

One of the caches maintained by **lookupd** holds the information used by the getpwent() library routine. This routine lists the users known to the system (see Computer Systems Research Group 1986 and the UNIX manual pages). **lookupd** can optionally cache the information it provides to getpwent(). The cache is enabled by default.

lookupd uses a ªlazy refreshº on this cache. By default, **lookupd** loads the cache when it starts, normally at boot time. It then refreshes the cache only if the cache is referenced. At regular intervals**, lookupd** checks to see whether the cache has been referenced, and reloads it if it has. In addition, if the cache is referenced after a periodic check, **lookupd** sends the old data to the caller, then refreshes the cache. (See the **lookupd**(8) UNIX manual page.)

To gain some perspective about this cache, consider that NEXTSTEP uses getpwent() only to complete a partial user name. For example, you can invoke this function in Workspace Manager's Finder by typing **~** followed by a partial user name, and pressing the Escape key. For example, if **smarco** is a user name, type ª~smarº, then press Escape. The Finder completes the name for you if the prefix you supplied is unique.

Logged-in user cache

The user information for the currently logged-in user and the **root** user are cached when someone logs in through **loginwindow**. **lookupd** refreshes this cache every 20 minutes and whenever the logged-in user changes his or her password. It clears the cache when the user logs out.

Local host information

The host informationÐhost name and Internet addressÐfor the local computer are cached when **lookupd** starts. If these change, **lookupd** must be restarted, typically

by rebooting the computer.

Printers

Information regarding available network printers is also cached by **lookupd**. Each time the printer database is accessed through **lookupd**, **lookupd** verifies the validity of the cache by comparing the current checksums of the source databases with the checksums from when the cache was last loaded. If the cache is out of date, **lookupd** reloads it and sends the new data to the caller. If the cache is current, **lookupd** just sends the data from the cache.

UNIX Groups

The cache for UNIX groups is maintained and consulted just like that for printers. It's referenced by the getgrent() library routine, but not by getgrnam() or getgrgid().

Other hosts

lookupd maintains a one-record cache for information about computers other than the local computer. The cache contains the most recently referenced host name and Internet address. **lookupd** accesses the cache in response to calls to gethostbyname(), but not for calls to gethostbyaddr().

Mount points

The cache for mount pointsÐlocations for imported file systemsÐis like the cache for printers. It's accessed for calls to getmntent().

LOOKING IT UP WITH LOOKUPD

Now let's examine what happens when an application makes a call to a library routine that provides system administration information. For this example, assume the application calls gethostbyname() to get information about a particular computer. The remote computer is **Tute.EDU** and is outside Rhino Aviation's network.

First, the application invokes gethostbyname(). This executes code in libc; the libc code checks to see if NetInfo is running. It is, so the libc code sends a Mach

message to **lookupd**, requesting that a gethostbyname() operation be performed.

When it receives the Mach message, **lookupd** first checks to see if the request is for information about the local computer. It isn't, so **lookupd** then checks to see if the last host name referenced was **Tute.EDU**. If it was, **lookupd** would return the information from the cache. For this example, though, let's assume some other computer was referenced last.

Next **lookupd** consults NetInfo to get the information. This results in an NI_LOOKUPREAD SunRPC message to the local NetInfo serverÐ**netinfod local**. Since the information isn't in NetInfo, this call returns an error and **lookupd** repeats the process, ascending the NetInfo domain hierarchy. This may require locating an appropriate NetInfo server, and so could require connecting or binding. (See ªNetInfo Binding and Connecting.º)

Since the information isn't in NetInfo and the application process is requesting machine-related information, eventually **lookupd** consults the DNS, using the normal resolver library routines. In this example, it finds the information in the DNS. It then returns the host information for **Tute.EDU**, using a Mach message, to the gethostbyname() libc routine. The libc routine then returns the information to the client application.

If the information hadn't been available from the DNS, then **lookupd** could have checked NIS, using the normal NIS semantics. At Rhino, though, NIS isn't used, so **lookupd** always stops with the DNS.

MANAGING LOOKUPD

Ordinarily, you may not have to deal with **lookupd** directly, so you don't need to do anything to manage it. However, if you want to know what **lookupd** is doing over time, it can log all requests it handles. You can also restart it if it's running into problems or if you need to refresh caches or tallies. The following sections explain how to work with **lookupd**.

Logging lookupd requests

Beginning in NEXTSTEP Release 3.0, **lookupd** can log information about requests

it receives. You can set the logging option for **lookupd** in the system startup script **/etc/rc**. Logging is described fully in the UNIX manual pages under **lookupd**(8).

For example, the arguments **-L** *file* cause **lookupd** to log information about requests it receives to the specified file. Logged information includes the called procedure, the number of calls to the procedure since **lookupd** started, the time required to process this request, and the total time consumed by all instances of this type of request. (Times are in microseconds.) When appropriate, the argument to the call is also logged, and the argument is prefixed with an asterisk if the data was retrieved from the cache. Figure 2 shows an example of **lookupd** logging output.

```
getservbyname             Ncalls: 1     Elapsed: 78    Total time: 78
gethostbyname (rhino)     Ncalls: 1     Elapsed: 32    Total time: 32
getservbyname (ntp)       Ncalls: 2     Elapsed: 108   Total time: 186
getmntent                 Ncalls: 1     Elapsed: 3384  Total time: 3384
gethostbyname (sabre)     Ncalls: 2     Elapsed: 18    Total time: 50
gethostbyname (ranger)    Ncalls: 3     Elapsed: 51    Total time: 101
gethostbyname (*ranger)   Ncalls: 4     Elapsed: 0     Total time: 101
setpwent                  Ncalls: 1     Elapsed: 24    Total time: 24
getpwnam (smarco)         Ncalls: 5     Elapsed: 28    Total time: 46
setloginuser (672)        Ncalls: 1     Elapsed: 64    Total time: 64
getpwnam (*smarco)        Ncalls: 6     Elapsed: 1     Total time: 47
getmntent (*)             Ncalls: 3     Elapsed: 39    Total time: 3461
gethostbyname (ranger)    Ncalls: 5     Elapsed: 38    Total time: 406
getpwnam (*smarco)        Ncalls: 7     Elapsed: 0     Total time: 47
getpwuid (*67)            Ncalls: 8     Elapsed: 1     Total time: 27
initgroups (smarco)       Ncalls: 1     Elapsed: 239   Total time: 239
getgrent                  Ncalls: 1     Elapsed: 1407  Total time: 1407
getpwuid (*0)             Ncalls: 9     Elapsed: 0     Total time: 41
getpwnam (*root)          Ncalls: 8     Elapsed: 0     Total time: 353
gethostbyaddr             Ncalls: 1     Elapsed: 32    Total time: 32
getpwuid (22)             Ncalls: 10    Elapsed: 27    Total time: 68
gethostbyaddr             Ncalls: 2     Elapsed: 12    Total time: 44
getservbyport             Ncalls: 1     Elapsed: 49    Total time: 49
```

Figure 2: *Logged information about lookupd*

Note that since **lookupd** is invoked by **/etc/rc** at system boot time, you have to modify **/etc/rc** and restart the system to enable logging. You can't turn logging on and off dynamically in either NEXTSTEP Release 3.0 or Release 3.1.

Restarting lookupd

Sometimes, you might want to restart **lookupd**. For example, you might want to force a cache to refresh, change the NetInfo servers being used, or reset the totals reported by the logging feature. You restart a **lookupd** daemon by sending it a ªhang-upº signal, also called a **SIGHUP**.

To do this, first find the **lookupd** process ID number, using **ps** for example. Then, run the following command as **root**, substituting the process ID number for *pid*:

```
kill -HUP pid
```

This kills and automatically restarts **lookupd**.

If you were instead to try to restart **lookupd** by terminating it and rerunning the program, your computer would hang, because the library functions would be unable to contact the new instance. (If you ever find yourself in this state, reboot the computer.)

Note: The document references in this and other articles in this issue refer to the books and articles listed in ªNEXTSTEP Networking References.º