Speeding Up UNIX Login by Caching the Initial Environment

Carl Hauser – Xerox Palo Alto Research Center

ABSTRACT

A package scheme helps users manage the environment variables needed by the applications that they use, but imposes a long delay during login while the environment is incrementally constructed. This paper describes an approach to caching the incrementally constructed environment. The mechanism caches different environments for different operating systems and is robust in the face of users' changes to their .login files. For the typical PARC user who enables 11 packages at login, caching reduces the time to login from about 30 seconds to about 5 seconds.

Introduction

The Xerox Palo Alto Research Center (PARC) research community and support staff use various UNIX systems and applications in the course of their daily work. Applications are stored on file servers and maintained for multiple systems using strategies similar to those used in the NIST Depot [2]. Each application is centrally maintained, but every user's computing environment is highly customized. Login scripts set environment variables, configure terminals, and so on, for the applications that the user actually uses. Managing the contents of the environment is burdensome for users who use many applications on many different kinds of systems.

To alleviate some of the burden on users, PARC implemented a Packages scheme similar to the Modules scheme described by Furlani [1]. Each application (actually application version) is stored on a file server in a directory tree structured according to the Packages conventions. The conventions require that every package's directory has a *top/* subdirectory containing a *README* file and C Shell scripts called *bringover* and *enable.* A user executes the *bringover* script once to establish the permanent state of the application in his home directory. Thereafter, the environment in any shell instance can be prepared for the application by executing the *enable* script. The *enable* script at least adds the package's *bin/* directory to the shell's *PATH* environment variable and its *man/* directory to the *MANPATH* environment variable. In general, however, *enable* scripts may affect environment variables in arbitrary ways.

Every user's *.cshrc* file defines aliases implementing *bringover* and *enable* commands taking a package name as an argument. The aliases locate the *top/* subdirectory for the named package, using the support programs of the Packages system, and *source* the appropriate OS-and-package-specific *bringover* or *enable* file. Therefore, before using a package the very first time, a user executes the command

```
bringover <packagename>
```

once. Thereafter, executing

```
enable <packagename>
```

in a shell instance sets up the environment variables in that shell to use the package. Users typically enable the packages that they use the most with commands in their *.login* files, but they also enable packages interactively, for example, to try out new software. Notice that, unlike Furlani's Modules, our Packages support only the C Shell (*csh*) and other shells that use the C Shell language. The caching techniques described here could be applied to module systems using other shells, but we have not needed to do so for the PARC environment.

Problem Statement

As was observed in the Modules system, a few seconds are needed to enable a package. This is acceptable when interactively enabling a single package, but it has proven unacceptable when many packages are enabled in the *.login* script. Since our researchers' work on interoperable and distributed systems leads them to login often to various machines, they soon find the delay during login becoming intolerable.

While reducing the time for each enable would be highly desirable, *csh*'s hashing of the directories on the search path with each change of the *PATH* variable imposes a lower bound at which the delay would still be too great. (The problem is compounded by a large directory containing standins for all known executables that many users place at the end of their search paths. The standins help users figure out what package needs to be enabled to provide the command, but with over 6000 entries the directory is very expensive to hash.)

Approach

Our caching approach is rooted in the observation that the state of the environment immediately after login is almost always the same for a given user on a given kind of machine. It would only be different if the user changed her *.login* file or the system administrators changed the effect of a package's *enable* script. Changing the *.login* file is an infrequent event. Changing an *enable* script is even rarer.

The environment caching mechanism described here reduces the delay during login by setting all environment variables exactly once during login from a file *source*'d from the user's home directory. A separate file is kept for each OS type. The absence of a cache file for an OS or a change to the *.login* file causes each package to be individually enabled so that the environment is correctly initialized. The cache is recomputed asynchronously at each login so there is at most a one-login delay in correcting the cache for a change made to an *enable* script. Thus, the existence of the cache is transparent to the user, excepting only the shorter time it takes to login and the potential to miss a (rare) *enable* script change for one login.

It would be possible to make the use of the cache sensitive to changes in the *enable* scripts by comparing a timestamp in the cache with the timestamp of the *enable* script, but this was rejected for three reasons. First, the additional time required to locate and *stat* the script files would slow down login in the most frequent case–that of no changes. Second, the vulnerability to changes would remain, because *enable* scripts may themselves have dependencies on other files that might change. Thus, users would have to be warned of the potential anomaly anyway. Finally, implementing such a test would further complicate the system. We judged that these negatives outweighed the benefit of a slightly more sensitive test for cache invalidity. For similar reasons, the benefits of accuracy and simplicity gained by completely recomputing the cache file at each login outweigh the reduction in system load that might be gained by trying to figure out when such a recomputation is really needed.

Implementation

Environment caching is implemented by a single, shared C Shell script *source*'d from users' *.login* files. To use it, users modify their *.login* files to initialize the shell variable *packages* with a list of the names of packages to be enabled and then *source* the file *.login-shared.* See Figure 1.

.login-shared provides the caching implementation. (The script appears in the Appendix should you want to follow along during the discussion.) It is invoked in two ways: as we have seen, it is *source*'d from users' *.login* files; and *.login-shared*, itself, invokes a *nice*'d, background, C shell also executing *.login-shared.* The first of these establishes the environment for the user's current login session, using a cache if one is available, while the other computes a new cache for use the next time the user logs in. (Separate script files implementing these two functions could be used, but having them in a single file is perhaps a bit easier on our administrators.) *.login-shared* determines which of these two things it's supposed to do based on the definedness and value of the environment variable *MAKEENABLECACHE:* if *MAKEENABLECACHE* is undefined, *.login-shared* must construct the user environment and build a new cache in the background; if *MAKEENABLECACHE* is **YES** it should build a new cache; and if *MAKEENABLECACHE* is **NO** it should do nothing (see discussion of *login -p* below).

To construct the user environment, *.login-shared* looks for a cache file (named *.login-enables-*

```
set path = ( /usr/ucb /bin /usr/bin $HOME/bin /etc /usr/etc \
    /usr/parc/bin )
set packages=\
    ( misc lemacs openwin X11R5 Xmisc afs gdb lcd )
if ( -r $HOME/.login-shared ) then
    # .login-shared enables each package listed in $packages
    source $HOME/.login-shared
else
    # normally done in .login-shared for fast enable cacheing
    foreach p ($packages)
        echo -n " $p"
        enable $p
    end
    echo ""
endif
```

Figure 1: Sample .login fragment

<platform> by default) and confirms that its *mtime* is later than that of the *.login* file. If so, it *source*'s the cache. As an additional consistency check, cache files are self-checking against the packages list that they implement. If all goes well, the cache file is *source*'d, constructs the environment and returns indicating success in a shell variable. Should either the *mtime* test or the packages list test fail, *.login-shared* takes the slow path of individually enabling each package in *packages.* Finally it sets *MAKEENABLECACHE* to **NO** and returns to the user's *.login* file.

The *.login-shared* instance that executes in the background receives the list of packages as its arguments. This shell sees a pristine environment in which none of the packages have been enabled. Its initial environment reflects only the contents of the user's *.cshrc* file and the *.login* file prior to its *source*'ing of *.login-shared. .login-shared* records this initial environment state in a temporary file and then enables each of its arguments. When it has finished all of them it compares the new environment with the old and produces a cache file containing a *setenv* command for each environment variable that changed or was newly defined. It is beyond the power of simple English to describe the *sed*, *sort*, *uniq*, and *awk* commands that accomplish the comparison between the results of the two *printenv* commands and their combination into a single collection of *setenv* commands, so please refer to the Appendix for the actual code. Figure 2 is an example cache file produced by *.login-shared.*[1]

One obvious thing to worry about is multiple logins occurring in close succession. Care is required to ensure that the new cache value is correct in this situation. Temporary file names that *.login-shared* creates include the process id of their creator. Furthermore, the cache files are created with temporary names prior to being *mv*'d to the proper place. Since *mv* isn't atomic, theoretically another login proceeding simultaneously could see an inconsistent state. However, should this happen, that login would just take the long path of separately enabling each package, so no real harm would be done.

The implementation supports all of the operating systems supported by the Packages system including Sun Solaris-1 and Solaris-2 for SPARC systems, IBM AIX 3.2 for the RS6000, SGI Irix 4 and Irix 5 for SGI systems, and OSF1 for the DEC Alpha.

Performance

A sample of 62 PARC Solaris-1 users enable between 3 and 26 packages in their *.login* files. The mean is 12 packages and the median and mode are each 11 packages. (Solaris-1 is the system used by a large majority of PARC UNIX users. One would be hard-pressed to find 26 enable-able packages for any of the other systems.) Recent measurements indicate that to enable 11 packages during login requires 28 to 35 seconds on a typical SparcStation 2 running Solaris-1. Using a cache to get the same effect takes 4 seconds.

Gotchas

The *.login-shared* file has gone through several releases over the last two years to correct deficiencies of the original design and implementation. Most have been to adapt the script to deal with the different locations of utilities such as *printenv*, *uniq* and *awk* on the various platforms we support.

[1] Figure 2 has been edited to reduce the line lengths. The setenv commands are not (and must not be) split over lines in the file.

```
#
if ( $?debugenables ) echo " "
if ( $?debugenables ) echo -n enable cache \
    created Thu Jan 27 14:47:32 PST 1994 by tregonsee
if ("$packages"==\
    "import-support-1.0 import-support gnu-2.0 sunpro bridge-2.0") \
    then \
        setenv MANPATH '/import/bridge-2.0/man:/local/sunpro/SUNWspro/man:\
/import/gnu-2.0/sparc-sun-solaris2/man:\
/import/import-support-1.0/sparc-sun-solaris2.2/man:/usr/share/man'
        setenv PATH '/import/bridge-2.0/p2:/local/sunpro/SUNWspro/bin:\
/import/gnu-2.0/sparc-sun-solaris2/bin:\
/import/import-support-1.0/sparc-sun-solaris2.2/bin:.:\
/sbin:/usr/sbin:/usr/bin:/etc:/usr/ccs/bin:/usr/ucb:/usr/openwin/bin'
        set didenables
endif
```

Figure 2: A small environment cache file

While tedious to correct, such problems are easily predicted in a multi-platform environment. Apart from the locations of the standard commands, no platform-specific customization has been needed, for example, to use different switches or different *awk* or *sed* scripts on the various platforms.

Two more subtle bugs have emerged and been corrected over this time. The first concerns explicit use of the the *login -p* command. (Recall that *login -p* passes its caller's environment to the login shell that it creates.) If *.login-shared* is invoked from a shell started with *login -p,* it must not compute a new cache based on the difference between the original environment it sees and the environment created by enabling the listed packages: the original environment already has the packages enabled. This is the purpose of setting *MAKEENABLECACHE* to **NO** in the environment. Since *MAKEENABLECACHE* is inherited by a forked login shell if other environment variables are, *.login-shared* recognizes the use of *login -p* and doesn't compute a new cache.

The other subtlety concerns environment values containing special characters. The *printenv* command does not quote the values in its output, so this has to be taken care of in the *awk* script that converts *printenv* output to *setenv* commands. While not difficult to fix, this bug was not triggered for a long time after the deployment of the caching programs.

Finally, while not directly a problem with environment caching, the improved performance of login has encouraged people to have lots of packages enabled. This has, in turn, pushed them up against *csh*'s 1K limit on the length of the search path. We have had to produce a variant of *csh* supporting paths up to 4K in length for use at PARC.

Conclusions

A package scheme providing scripts for setting up shell environments can be very useful to a large community using many applications, but both the Modules system and the PARC Package system suffer from the long time it takes to establish the initial environment at login. The environment caching technique described here reduces the time taken by login by about 25 seconds for a typical user at PARC. Since logins are usually not easily overlapped with other work activity, they tend to be particularly disruptive to thought processes. Saving 25 seconds here is seen as more valuable than saving 25 seconds in some other contexts.

The cache validation scheme used is robust enough to immediately implement changes that a user might make to her list of enabled packages, but lags by one login changes that administrators might make to the effect of a package's *enable* script. Most users have found this behavior acceptable. Users who are uncomfortable with this behavior can easily opt out of using the caches by enabling packages directly in their *.login* files and invoking *.login-shared* without setting *packages.*

Acknowledgements

PARC's Package system was originally conceived and implemented by Stan Lanning for SunOS 4.1. Jim Foote contributed much of the multiplatform capability. Dale MacDonald currently maintains the Package system and many of the most commonly used packages. Steve Putz provided examples and fixes for the problem of environment values containing special characters, and Mark Verber acquainted me with the Modules work. As always, discussions with Al Demers provided new insights.

Availability

.login-shared is available for anonymous ftp. The URL is file://ftp.sage.usenix.org/pub/lisa/lisa8 /hauser.tar.Z

Author Information

Carl Hauser joined the Computer Science Laboratory at the Xerox Palo Alto Research Center ten years ago as a Member of the Research Staff after five years at the IBM San Jose Research Laboratory. He develops language run-time implementations for multi-threaded languages and has a particular affinity for developing caching solutions to performance problems. His 1980 dissertation at Cornell University concerned verification of parallel programs. Reach him by mail at Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 or via electronic mail at the address chauser@parc.xerox.com

References

[1] Furlani, J. ''Modules: Providing a Flexible User Environment.'' USENIX Large Installation System Administration V Conf. Proceedings, 1991, pp. 141-152. URL: file://ftp.sage.usenix.org /pub/lisa/lisa5/furlani91-modules.ps

[2] Manheimer, K., Warsaw, B., Clark, S., and Rowe, W. ''The Depot: A Framework for Sharing Software Installation Across Organization and UNIX Platform Boundaries.'' USENIX Large Installation System Administration IV Conference Proceedings, 1991, pp. 37-76. The URL is file://ftp.sage.usenix.org/pub/lisa/lisa4 /manheimer90-depot.troff

Appendix A: .login-shared

```
#!/bin/csh -f

# .login-shared: Usage in a .login file

# if you want simple enable caching:

#   set packages=(blank-separated-list-of-package-names-to-be-enabled)
#   # if you want enabling to proceed silently define silentenabling
#   #set silentenables=yes
#   if ( -r $HOME/.login-shared ) then
#      # .login-shared enables each package listed in $packages
#      source $HOME/.login-shared
#   else
#      # if things are set up right, this branch should never be executed
#      foreach p ($packages)
#           enable $p
#      end
#   endif

# if you enable different packages in different situations

#   if (situation-1) then
#      set cachename=.login-cache1
#      set packages=(list-of-packages-for-situation-1)
#   else if (situation-2) then
#      set cachename=.login-cache2
#      set packages=(list-of-packages-for-situation-2)
#   else if ...
#   endif
#   if ( -r $HOME/.login-shared ) then
#      # .login-shared enables each package listed in $packages
#      source $HOME/.login-shared
#   else
#      # if things are set up right, this branch should never be executed
#      foreach p ($packages)
#           enable $p
#      end
#   endif

# if you don't want enable caching

#   unset packages
#   if ( -r $HOME/.login-shared ) then
#      # .login-shared enables each package listed in $packages
#      source $HOME/.login-shared
#   endif

# That's the end of the documentation for ordinary users.  What
# follows is detailed documentation of the caching system.

# Create a .login-enables file to be sourced by .login.  The produced
# file contains setenv commands that reflect environment changes made
# by enable files for the listed packages.  The advantage of using
# enable caching is that the environment setting during .login
# processing goes fast, while the hard work of figuring out what the
# enable files do is deferred.  The disadvantage is that the
# environment variables will be set according to the enable files as
# they existed at the previous login--which, it is claimed, is not too
# bad since enable files are slowly changing objects.

# This file is processed twice: once when sourced from .login, once in
# a forked csh. The environment variable MAKEENABLECACHE controls its
# operation.
```

```
# MAKEENABLECACHE conventions
#  == YES: processing in a forked csh; .login-enables should be created.
#  == NO: processing was sourced from a .login that itself is
#     executing in an environment where enables have already been done.
#  unset: processing was sourced from a .login that itself is
#     executing in an environment where enables have not yet been done.

# To start using enable caching, change your .login file as described
# above; each subsequent login will use an existing .login-enables to
# speed enable processing then create a more up-to-date one for the
# next login.

# Bugs:
#   SunOS4.1.1 dependent?
#   mv is not really atomic.
#   Presumes that the only effect of the enable files is to modify the
#       values of environment variables.

# Don't alter the search patch set by .login.  Instead, explicitly
# reference the desired utilities depending on the host platform:
set platform = `/import/import-support-1.0/bin/sys-os-type.1`

if ( ! $?MAKEENABLECACHE ) then
   if ( ! $?packages ) then
    # If $packages is not set, .login has not been set up to use
    # this mechanism properly.
    # We might want to give some advice about using caching enables
    # here.
      exit
   else
   # enable import-support-1.0
   set packages = (import-support-1.0 $packages)
   # if ( $?packages ) then
     # No MAKEENABLECACHE in environment so fork self to make the file.
       if ( ! $?cachename ) set cachename = .login-enables
       setenv ENABLECACHE $HOME/$cachename-$platform
       setenv MAKEENABLECACHE YES
     # Putting the following command in "()"s make messages go to /dev/null
     # instead of cluttering up the console.
     # But first, sleep a bit so as not to get in the way of the login.
       (sleep 15; /bin/nice /bin/csh -f $HOME/.login-shared $packages &)
     # Henceforth, logins that inherit the current environment should
     # not do this again.
     # If there was no .login-enables for this platform, or if the
     # .login-enables is older than the .login file, we're forced to
     # do the enables synchronously
       unset didenables
       if ( -r $ENABLECACHE ) then
          set LSRESULT = `/bin/ls -c -t $HOME/.login $ENABLECACHE`
          if ("$LSRESULT[1]" == "$ENABLECACHE" ) then
             # sets didenables if the package list matches
             if ( ! $?silentenables )  echo -n "fast enable: $packages"
             source $ENABLECACHE
             if ( ! $?silentenables && ! $?didenables) echo -n \
               " -- failed; maybe the list changed?"
             echo ""
          endif
          unset LSRESULT
       endif
       if ( ! $?didenables ) then
          if ( ! $?silentenables )  echo -n "enabling:"
          foreach p ($packages)
```

```
              if ( ! $?silentenables )  echo -n " $p"
              enable $p
           end
           if ( ! $?silentenables )  echo ""
        endif
        unset didenables
        unsetenv ENABLECACHE
     # endif
     setenv MAKEENABLECACHE NO
    endif # $?packages = T
 else if ( $MAKEENABLECACHE == YES ) then
   # This is the forked self.  Update the compiled enable file
   # $ENABLECACHE.
   # Make the compiled file self-validating wrt the package list
     echo "#" > $ENABLECACHE.$$
     echo 'if ( $?debugenables )' echo '" "' >> $ENABLECACHE.$$
     echo 'if ( $?debugenables )' echo -n 'enable cache created' \
          `date` by `hostname` >> $ENABLECACHE.$$
     echo 'if ( ' \"\$packages\" " == " \"$*\" ' ) then '\
           >> $ENABLECACHE.$$
   # Capture the current environment.
   # Each line in prefaced with a "b=" to mark it as being
   # "before" the enables.
     switch ( $platform )
           case mips-sgi-irix4:
           case mips-sgi-irix5:
           case alpha-dec-osf1:
           case rs6000-ibm-aix:
               set PRINTENV = /bin/printenv
                 breaksw
           case sparc-sun-solaris1:
           case m68k-sun-solaris1:
           case sparc-sun-solaris2.3:
           case i486-sun-solaris2:
           default:
               set PRINTENV = /usr/ucb/printenv
                 breaksw
     endsw
     $PRINTENV | /bin/sed -e "s/^/b=/" > /tmp/environ$$
   # our own definition of enable since user's .cshrc may not execute
   # .cshrc-shared in this forked process
 # n.b. for LISA VIII readers:
 # the following command had to be improperly split
 # across lines to fit on paper.  Retrieve the
 # actual script to assure correctness.
      alias enable \
          'source "`/import/import-support-1.0/'$platform'\
               /bin/package-file-name enable \!*`"'
   # Do an enable for each argument.
     foreach p ($*)
        enable $p
     end
   # Capture the resulting environment, adding it to the file created
   # before the enables.
   # Each line is prefaced with a "a=" to mark it as being
   # "after" the enables.
     $PRINTENV | /bin/sed -e "s/^/a=/" >> /tmp/environ$$
     unset PRINTENV
   # Sort the file with all the environment values.  The sort is
   # carefully designed to bring together lines that define the same
```

```
  # environment variable, with the "after" line before the "before"
  # line (if there was indeed a "before" value).  Note the clever,
  # indirect use of the "a=" and "b=" that were added to each line --
  # we depend on the fact that "a" comes before "b", and that "=" is
  # the field delimiter.
    /bin/sort -t= +1 -2 -o /tmp/environ$$ /tmp/environ$$
  # Delete things that didn't change at all.  Note how the leading
  # "a=" or "b=" are ignored in the comparison -- we depend on the
  # fact that the "a=" and "b=" were added to the beginning of the
  # line.
    set UNIQ = /bin/uniq
    if ( $platform == mips-sgi-irix4) then
      set UNIQ = /usr/bin/uniq
    endif
    if ( $platform == mips-sgi-irix5) then
      set UNIQ = /usr/bin/uniq
    endif
    if ( $platform == sparc-sun-solaris1 ) then
    # solaris-1 uniq is (silently) broken on lines longer than
    # 1000 characters
      set UNIQ = /import/textutils/sparc-sun-sunos4.1/bin/uniq
    endif
    $UNIQ -u +2 /tmp/environ$$ /tmp/environ.uniq$$
    unset UNIQ
  # Collect the lines that remain and that come from the "after"
  # environment.  At the same time, convert the syntax of the lines to
  # "setenv" commands.  If I were really an awk hacker, I could
  # probably have this command do the "uniq" stuff above, too.  But
  # I'm not.
    set AWK = /bin/awk
    if ( $platform == mips-sgi-irix4) then
      set AWK = /usr/bin/awk
    endif
    if ( $platform == mips-sgi-irix5) then
      set AWK = /usr/bin/awk
    endif
    /bin/sed "s/'/'\\''/g" /tmp/environ.uniq$$ \
        | $AWK -F= 'BEGIN {sq = sprintf("%c", 39)} \
        $1 == "a" { print "  setenv " $2 " " \
          sq substr($0,length($2)+4) sq }' \
        >> $ENABLECACHE.$$
      unset AWK
      echo "  set didenables" >> $ENABLECACHE.$$
      echo "endif" >> $ENABLECACHE.$$
  # As atomically as possible, move the uniquely-named file to the
  # standard place.
    /bin/mv -f $ENABLECACHE.$$ $ENABLECACHE
  # Remove the temporary files.
    /bin/rm /tmp/environ*$$
endif # $MAKEENABLECACHE = YES
  unset platform
```