Autoconf for large scale portability: Kerberos V5, a case study

Mark Eichin

Cygnus Support
Somerville, MA 02144
`eichin@cygnus.com`

Abstract

In 1994, Cygnus switched the entire Kerberos V5 development tree from Imake over to autoconf. The effort was driven by difficulties with Imake, and projected advantages of autoconf. The effort is considered a success, although it has introduced new challenges which are still being dealt with.

1 Introduction

The MIT Athena environment, the "prime customer" for Kerberos, supports from 3 to 5 current operating systems at any given time, making simple multiple platform support mandatory. Since the success of Kerberos depends somewhat on wide scale adoption, broad portability is even more important.

The Kerberos V5 sources are currently a collection of 2200 files spread over 150 directories. A build produces over 50 executable programs and a dozen libraries. Two years ago, it had been managed by the use of "Imake", the tool used to generate the makefiles for X11.

Imake has a number of flaws, and some of them were interfering with the effort to port Kerberos V5 to different platforms. These include

- Imake options are compiler-specific
- Imake makes poor use of `cpp`
- Imake is tuned for simple X programs
- Imake is hard to retune for other use
- Hard to make minor adjustments

Autoconf had a number of predicted advantages, including

- Easy per-directory customization
- Feature testing - enhances portability of code
- Extensibility - makes new work easier
- Based on portable tools

We learned a number of things during and after the conversion that made it all worth the effort.

1.1 Imake problems

Shortly after Cygnus started serious work on V4, we started running in to problems with imake.

1.1.1 where imake succeeds

Imake encapsulates a standard set of operations that are performed in the normal X11 build cycle, in a way that makes it easy to either drop a program into the X11 build tree or to build it against an existing X11 installation. Definitions are provided for "reasonable" compiler options, installation path names, dependencies on libraries, and similar options.

If a vendor has correctly installed and configured Imake, then the Imake config files can be a good source of information about the system. Various non-X11 packages including `tcsh` and Kerberos have made good use of this information in the past, though it is most effective when used to configure X11 applications.

1.1.2 where imake fails

Imake options are compiler-specific For example, X11R6 under HP/UX has options for the unbundled HP compiler, which don't quite work with the bundled one, and don't work at all with gcc. Most configurations have some provision for running imake with the `HasGcc2` flag set... which sometimes has problems, and ignores other compilers altogether.

Imake makes poor use of cpp Imake depends on the behaviour of a berkeley style, non-ANSI C preprocessor. (It was written, after all, for the Vax BSD cpp in particular. [ref?]) Thus one needs to avoid contractions in Makefile comments, since cpp may see them as unterminated character constants.

Imake is tuned for simple X programs The standard Imake templates work best when using SimpleProgramTarget to produce single executables. There are more sophisticated rules, which do handle packages of similar structure to the X11 release, but aren't otherwise very flexible.

Imake is hard to retune for other use In order to make any changes to the X11 configuration, you either need a fairly complex set of definitions in every Imakefile, or you need to replace the Imake template altogether. Kerberos V4 and V5 both did the latter – adding even more distance from the advantages imake would otherwise provide.

Hard to make minor adjustments In the HP/UX example above, the only real workaround is to replace the templates. Solaris has also shipped (2.3) with broken templates; often the easiest approach is to install X11R6 directly, but this loses many advantages of even having the vendor provide it in the first place.

1.2 autoconf advantages

1.2.1 Easy per-directory customization

autoconf-generated "configure" scripts can check for command line options, perform a set of tests, and then produce a Makefile or set of Makefiles. The script can then look in a set of subdirectories for other configure scripts that need to be processed. (Imake only supports the latter mode.)

1.2.2 Feature testing - enhances portability of code

The primary appeal of autoconf is that you can specify in a high level manner that you intend to use some feature of the target system. Autoconf then produces a configure script which explicitly tests to see if the feature is available.

The scripts report their progress, which makes it easier to verify new tests or to see what types of features a particular system actually provided.

1.2.3 extensibility - makes new work easier

Since the tests are designed to check behavior, it is easy to tell how a new, unusual ("creatively wrong"?) behaviour can be identified, if it doesn't cause the existing test to fail. For example, regcomp() on Solaris 2.4 always fails, even for simple patterns. This can be expressed as a test which tries to actually use regcomp; by avoiding any dependency on the identity of the system, if Solaris 2.5 fixes the problem [does it?] then the test will not need any change to use the function.

1.2.4 Based on portable tools

Autoconf scripts are simply M4 macro packages. M4 has been a standard macro preprocessor on Unix systems for decades [ref?] It has suffered from two main limits - most implementations (before the GNU version) had fixed length buffers that drastically limited the complexity of macro expansions, and it couldn't replace CPP, because system headers were always supplied in CPP form (now crystallized in ANSI and ISO standards.) autoconf makes use of the added flexibility of GNU m4, and the sophistication of the m4 macro language, to provide clean "high level" encapsulations of configuration tests.

2 experience

2.1 what worked

The advantages described above did in fact pay off. Kerberos V5 has been ported successfully to a number of new platforms and new variations of operating systems, with less effort than we would have expected from an Imake based system. (We can confirm this partially by the effort needed to maintain the kerberized `xdm` program, which extracts information from Imake on the platforms it has been ported to.)

2.2 Surprise Advantages

Several things turned up as a result of the conversion effort that were unexpected benefits.

2.2.1 changes the porting question

By using feature-oriented tests, it was often much clearer when a feature might *not* be portable. This provided some encouragement to rewrite code to use consistently available features. In other words, try to be more portable in the first place.

While the Kerberos V5 code makes a fair effort to support non-ANSI compilers (such as SunOS or Ultrix native "cc") and non-POSIX environments (such as those lacking sigaction) we've found it makes the code significantly more complex, and plan to change this in the future (as does the MIT Zephyr project.)

One of the software engineering advantages provided by Ada95 is a standard set of packages. Assuming ANSI/POSIX isn't quite as good as that, but it gets a lot closer.

2.3 what would we do differently

2.3.1 be more aggressive in designing new macros

In the early stages of the project, we treated autoconf as a fixed package of useful tests. As we found the need to extend the "vocabulary" of features, when we couldn't get something to fit within the existing framework we reluctantly added other tests. Likewise, we often added features to local configure.in files that would have been useful across the whole project. Usually these got copied and then unified; sometimes they were duplications of effort that would need more work to unify.

More recently, we've discovered (as we became more proficient at it) that writing higher level macros was very rewarding in terms of simplification of future effort and in making configuration scripts more descriptive and more consistent. This also means we can fix things in one place.

2.3.2 be more aggressive about switching directory structure

The overall layout of the V5 tree hasn't changed since the early days of the project. This has actually reduced sharing of code in some ways; having one program per subdirectory was useful in the days of the Imake "SimpleProgramTarget" macro, but now the ability to share more code can be taken advantage of.

2.3.3 push harder for integrating new features

Many of the macros we've written for the V5 configuration have been fed back to the FSF and integrated into later autoconf releases. This has been especially the case when we made improvements to existing functions. We have not worked hard enough at providing our other developments in a form that would benefit others, and should remedy that; our goal has been to have only a few v5-specific items in the top level aclocal.m4 and to have everything else available in the real autoconf release.

3 future issues

3.1 style guide

Training turned out to be a big issue; there are still a few "gurus" doing kerberos v5 work that get consulted on changes or new configuration files. A style guide should reduce this significantly. Still, many packages included in the kerberos tree have similarities that have let people easily duplicate existing configuration files or fragments to get the effects they need. They end up missing a lot of opportunities to reduce maintenance by coalescing commonly repeated actions into higher level macros, something that has not been emphasized enough in our development.

3.2 performance

One outstanding complaint about configure is the time it takes to run the configure scripts. A few improvements such as caching test results have helped, but it still seems to take longer than it should... one quantitative expression of this is that one some systems, it takes longer to configure than it does to compile. [charts to be included later]

A number of experiments have determined that on most platforms, there isn't one overall slow point. An autoconf generated configure script does do a lot of forking, but a number of theories [look them up!] as to what else might be a bottleneck have not panend out.

The current approach being taken in kerberos v5 development is to attempt to reduce the amount of configuration done, by doing common testing in one place. This seems the most likely to reduce the total amount of forking due to individual programs, as well as reducing the maintenance of duplicated code.

4 what do we mean by portability

Why is portability especially important for free software?

- Free or university projects are often low on resources, and yet have the goal of producing tools that can be used on any available hardware. In particular, supporting future platforms with minimal long term resources.

- free software works best when cooperative work is possible; having most of the large-scale work of porting to a "new" system handled automatically is of great value in getting such help.

5 what we did to autoconf for kerberos

5.1 common file support

autoconf mostly supported figuring out features on a directory by directory basis. However, the Imake influence on previous V5 work meant that most Makefiles had large similar sections. Support was added to autoconf to allow a common prefix and postfix fragment to be attached to each makefile and processed.

6 Other Effects

6.1 configure scripts as data

In the abstract, it makes sense to think of the autoconf input scripts as high level descriptions of what a platform needs. In practice, it also works. Since it's easy enough to port simple programs that only use ANSI (POSIX?) file I/O to the mac, a mac port of m4 and a new set of macros was used in one porting effort.

6.2 cross-compiling works

Since all but a handful of the tests only need a compiler or linker, and don't actually need to execute, it is straightforward to port to a new platform for which a cross compiler exists.

Autoconf (and Kerberos) would seem rather Unix-specific. However, we completed a port of all client libraries and a number of client programs to the win32 ABI using the Cygnus win32 cross-development tools and support library. Most autoconf tests only require the ability to run tools (compiler and sometimes linker) rather than executing programs; this means that a cross-compilation environment can be used to perform the autoconf tests. What few tests couldn't be handled this way (a total of 8 tests in the entire V5 tree, some of which could be easily fixed) can be handled by entering the answers directly into the config.cache.

It took a total of three weeks from the initial installation of NT to a successful test of a kerberized telnet session; most of the work was actually in improvements to the "winsup" library which provided unix-like interfaces to some low-level win32 functionality.

6.3 how to *use* autoconf on a package this big

While this doesn't replace a complete style guide, some examples of what we found to work(?) might be useful.

The autoconf manual identifies features likely to be at issue. Many predefined macros like `AC_CHECK_HEADER` or `AC_REPLACE_FUNCS` are easy to drop in as replacements for existing ifdefs, especially if they are already feature-specific.

Common header file vs. local in-makefile tests autoconf can provide the results of a set of tests in two primary ways. The simplest way is to simply drop -D directives directly into the generated Makefile. This is widely used in the kerberos tree, but means configuring every directory for a particular set of tests. This can take a great deal of time, and generally duplicates a fair amount of work.

The other approach is to determine all of the features you intend to test and generate a single master header file with the results. Sometimes it makes sense to simply handle the inclusion of varying system headers here directly. While this means that configuration is done faster in the normal case, it can slow down localized development by pushing all testing into the common directory, leading to merging and total-rebuild overhead.

A hybrid mechanism would probably be useful, but requires more time be spent surveying the options actually used.

program-per-directory vs. collections of programs Kerberos V5 has a number of single-program directories. This was mostly driven by the availabilty of the Imake "SimpleProgramTarget" mechanism, and kept individual programs together with their own documentation.

Other directories (such as appl/bsd, carried over from V4) are sets of programs that share a fair amount of common code, but perhaps not enough for a library.

6.4 ease of porting

"Porting" from solaris 2.4 to solaris 2.5 shouldn't be a major effort, yet there are usually significant changes, and software such as Kerberos which is to some extent "system" software is more sensitive than most to such changes. Having feature oriented tests saved a great deal of effort.

autoconf feature-based tests also simplified porting to totally new platforms, such as HP/UX, because we only needed to locate where features were assumed and not tested, or where the tests were inadequate. In most cases the tests performed correctly, covering the bulk of the "easy" work in the port, and letting the developers get right to the challenges.

6.5 conversion issues

The possibility of using M4 macros to directly emulate the Imake macros was considered, and in fact some functions (CopyHeaderFile and the like) still exist. There is a question of style in pushing "actions" into the configure.in file, though it can be done easily enough; the Makefile.in files could be eliminated altogether. This was seen as being "too abstract" and would make it more difficult to change the build system in the future.

Note that in the original port, while some of the work was done based on the Imakefiles that were in place, the rules that went into the Makefile.in files were often taken from the imake-generated Makefiles directly, but made more directly portable.

emacs works in spite of it, using xmkmf As discussed, Imake is still the primary source of installation-specific information about X11. Rather than duplicate all of that effort, there is autoconf support for running xmkmf on a small Imakefile and then parsing that to find out the particular variables we need. Emacs uses this approach

appl/xdm similar The xdm build in kerberos goes a little farther, since it needs some variables (ConnectionFlags in particular) that are only exported through imake, rather than through header files.

6.6 mistakes we made, plans to fix them

not training others more? Some amount of confusion has resulted from developers learning *only* from the examples in the tree [see TrainingTrainers fallacy]

not writing a style guide A style guide provides a fair amount of history, instead of having any individual developer be the repository of knowledge about the reasons things were done.

6.7 things we didn't try that might have made it easier

We should have done even *more* to produce new functional units There is a fair amount of duplicated testing code in the tree, sometimes because developers didn't know about available cases in autoconf, sometimes because the cases were added to the autoconf release later, and sometimes because the solutions were in other unfamiliar parts of the tree. Time will be spent going through and collecting useful idioms into verbs, something m4 is well suited to.

tried recognizing similar directories (like ken did to clients) As part of the hybrid effort mentioned earlier(?) since there are already some groupings of related programs, having the tests for those programs done once and then propagated directly into Makefiles has successfully reduced the amount of time spent configuring the tree.

6.7.1 "amazing fringe benefits"

Features like –prefix and –srcdir came nearly for free, and greatly simplified both building and installing the system. Other mechanisms, such as –enable-* found application as we became more familiar with autoconf's abilities and found that we could provide options; we may have gone a bit too far overboard in that effort, however, in making the package too configurable and thus harder to test.

7 Alternatives

There are other software construction systems which we did not use.

7.1 metaconfig

"metaconfig" is mostly focussed on determining attributes of the system that are best handled in more portable ways. It also asks the user for confirmation of questions the user is unlikely to know. Though this is useful for helping a developer, the autoconf approach provides just as much information without scaring the end user. It also seems tuned for single large packages (perl, rn) and as such has weaker support for systems.

7.2 Cygnus configure

Kerberos V4 also started as an imake-based system, but the Cygnus maintenance effort was hampered by the difficulty of finding a cpp that worked with imake and was portable to AIX (among other platforms.) Inspite of the intent, the preprocessing that Imake assumed was *not* reliably available from the native compiler, due to pre-ANSI assumptions.

Cygnus already used a common "configure" script to build a collection of GNU tools together. The approach seemed to fit well with the smaller Kerberos V4 system, which had some related properties: libraries built with in the tree, collections of applications that used those libraries split up in different ways for different reasons.

CNS V4 switched to using "Cygnus Configure", which doesn't do any tests but has a simpler way of managing per-platform configuration information as makefile fragments. Since we already had platform-specific information from the Imake config files, this was simply a better way to organize it. Adding new platforms that have only minor differences is straightforward – add a pattern to match the `config.guess` output, and modify some existing makefile fragment with the appropriate changes.

"configure" is still a hard-coded porting system; it identifies a given platform, based mostly on "uname" output and then uses pattern matching on that string to select makefile fragments appropriate to the platform. This means that porting is sometimes a matter of creating a new fragment by copying parts of fragments from similar ports; for example, the Solaris port started from a copy of the unixware svr4 configuration.

8 Conclusion

The effort expended in switching over to autoconf has been recovered in portability improvement. There are still areas where we could make better use of autoconf, or use it more efficiently, and further work is being done in that area. However, none of the developers would consider going back.

A Why not use Autoconf for X11?

Producing an X11 release that used autoconf would be of similar scale to the whole kerberos v5 autoconfiscation effort, and would really need buy-in from the X Consortium to be worth maintaining. If it weren't for the wide base of imake-using applications, it would be straightforward.

A.1 load system-wide config.cache

To avoid actually running the X-specific tests every time, a system-wide config.cache could be provided. Note that these should really be compiler-independent features, or autoconf would need to be extended to parameterize some cache values in more detail than it does.

A.2 drag lots of examples over

In order to help convert existing uses of Imake, there are a few helpful tricks. One would be a set of m4 macros that actually implemented most of the Imake primitives... then a small amount of text massaging would be enough to directly translate Imakefiles. Another help would be to provide plenty of examples (in the form of the X release itself, and the Imake examples from published sources) that show how easy the conversion is, to guide others in doing it manually (and gaining benefits of additional portability.)