buildlink3: Methodology and Philosophy

Johnny Lam
jlam@NetBSD.org

Outline

- Problem description
- Short history of different solutions in pkgsrc
- Unfinished work/ideas
- Summary

The problem of repeatable builds

- How to do repeatable builds of packages regardless of the order that the packages are built?
  - E.g. PostgreSQL can build optional server-side language modules if Tcl/Tk and/or Perl are installed.
    - If we build on a clean system, the resulting PostgreSQL binary package doesn't have any dependencies.
    - If we install Tcl/Tk and Perl, then build PostgreSQL, the binary package depends on Tcl/Tk and Perl.
    - PLIST is different between the two builds
- We want to precisely control what dependencies a package can have. Basically, we want to tell the build process which packages we want as dependencies and ignore everything else that's also installed on the system.

1st try: chroot sandbox

- Use `chroot(8)` to duplicate a full install of the operating system, add some binary packages for dependencies, and build the package.
  - This is the basic idea behind `pkgtools/pkg_comp`.
    - A lot of the stuff that the buildlink3 framework does isn't necessary if you use `pkg_comp`, but we don't take advantage of this.
- Pros:
  - Perfect control over the build environment – if it's not in the chroot, then it can't be found.
- Cons:
  - "Massive" use of disk space. My Macintosh LC III running NetBSD-1.1/mac68k devoted half its disk space to just the base OS. Not enough room left over to install another copy of the OS into a chroot.
- Approach abandoned since it couldn't work on my machine.

2nd try: buildlink1

- Symlink headers and libraries into a directory and make the build look for those files inside that directory before `/usr/pkg` by passing appropriate `-I` and `-L` options to the compiler/linker.
- "build" the package against the sym"link"s, hence "buildlink" (aren't I clever?)
- Each package that supplies headers and/or libraries has a `buildlink.mk` file that lists the files to symlink into the buildlink directory (`BUILDLINK_FILES`)
- Pros:
  - Easy to tell the compiler to look for files in the buildlink directory before looking anywhere else, a.k.a. "weakly buildlinked"
  - Symlinks take up practically no disk space

2nd try: buildlink1 (cont.)

- Cons:
  - Had to read through and patch configure scripts and Makefiles to make the build not look outside of the buildlink directory, a.k.a. “strongly buildlinked”
    - Very time-consuming process.
    - A lot of up-front work at pre-configure time.
    - Easy for GNU software, but nearly impossible for software that used `imake` without heavily editing the `imake` config files
  - Had to remove references to the buildlink directory in installed files, e.g. GNOME `*-config` scripts, libtool archives.
    - Had to be vigilant that all references were purged. Very often, some files were overlooked.
  - Couldn't symlink a library into the buildlink directory with a different name, e.g. pretend `/usr/lib/libcurses.so` was really ncurses, since it broke when linking on a.out
  - Resulting package Makefiles were much more complex after conversion.

3rd try: buildlink2

- Keep the working idea of symlinking headers and libraries into a buildlink directory
- Packages have `buildlink2.mk` files that list the files to symlink
- Instead of directly invoking the compiler/linker, use wrapper scripts
  - Transform `/usr/pkg` into the buildlink directory
  - Ignore stuff in `/usr/local` and `/usr`

3rd try: buildlink2 (cont.)

- Pros:
  - Package Makefiles were simple again.
  - Didn't have to edit GNU configure scripts any more since the configure script thinks it's using files in `/usr/pkg` but it's really using files in the buildlink directory.
  - Could pretend a library had a different name, e.g. tell the wrapper to link against `-lncurses` and actually link against `-lcurses`.
  - Worked with X11 packages that used `imake` – no more "weakly buildlinked" packages.
  - libtool wrapper script automatically fixed up libtool archives for us.
  - The wrapper scripts could be used to fix problems with compilers on non-NetBSD systems
    - pkgsrc started being ported to Solaris and Linux around this time. Later, Darwin joined the cast.
  - Discovered the buildlink technique was portable across many different OSes.

3rd try: buildlink2 (cont.)

- Cons:
  - Build took longer than before due to overhead of transformations in the wrapper scripts.
  - Wrapper scripts weren't originally designed to help make pkgsrc more portable, so scripts grew crufty.
  - Only ignored stuff in `/usr`, `/usr/local`, and `/usr/pkg`, but allowed linking against libraries outside of those directory trees, e.g. `/home/oracle`
  - `buildlink2.mk` files for packages that duplicated software in the base OS sometimes needed to create fake libtool archives
    - This often broke for OSes with native pthread libraries
  - Didn't work with package views
  - `buildlink2.mk` files forced recursive dependencies

last try: buildlink3

- Perfection! (for some value of “perfect”)
- Redesigned wrapper scripts to be easier to port to different OSes
  - Can customize wrappers for different compilers
- No longer symlink stuff in `/usr/{include,lib}` into the buildlink directory – just use them where they lie
- Packages have `buildlink3.mk` files that tell pkgsrc about where the actual headers and libraries are found
- Packages have `builtin.mk` files that encapsulate the complexities of dealing with packages that duplicate software in the base system
- Designed from the start to integrate with the package views implementation

last try: buildlink3 (cont.)

- Pros:
  - No longer need to create fake libtool archives
    - buildlink3 is smarter about munging libtool archives in the buildlink directory
    - Solved large number of PRs related to libpthread, gettext-lib and libiconv
  - Wrapper scripts can make other compilers look and behave like GCC.
    - No need to add extra code/patches to packages to use different compilers.
  - `buildlink3.mk` files are easier to maintain
    - Don't have any code to deal with built-in software
    - Don't need to list files to symlink anymore – `bsd.buildlink3.mk` figures it out automatically by examining the installed package
  - No longer forces recursive dependencies (via `BUILDLINK_DEPTH`)

Final frontier

- Problem: GNU configure scripts often test for the presence of `*-config` scripts and other executables in the `PATH` by trying to execute them.
  - Often need to tune `CONFIGURE_ENV` to avoid finding random executables
    - e.g. Add `GLIB_CONFIG=no` to `CONFIGURE_ENV` in the package Makefile to avoid finding glib, even though we don't include `glib/buildlink3.mk`.
- Solution?
  - Ignore the `PATH` passed in from the environment and set a `PATH` used by the build that excludes everything except binaries in the base OS in `/bin`, `/sbin`, `/usr/bin`, etc. and binaries under `${WRKDIR}`.
  - Teach buildlink3 to also symlink `*-config` scripts into the buildlink directory
  - Not yet implemented, but discussed with `jmmv@NetBSD.org`

Summary

- Each iteration of buildlink does a more thorough job of hiding everything except the files that you explicitly say you want
  - We're basically emulating a chroot build by using shell wrapper script trickery that's portable across OSes
- Every future modification of the buildlink3 framework should be judged against this ideal
  - Changes that take us farther away should be rethought or rejected
  - Changes that take us closer should be cleaned up and incorporated