Can I haz debug

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>

Epita Research

December 10, 2019

Serendipity in Bucharest

The story so far

We had a ports hackathon in Bucharest, and during the first evening at dinner, two groups of people talked about debug packages

A tale of two developers

Paul Irofti

- Figured out the debugger part
- A recent gdb (gdb from ports) can grab debug info elsewhere
- And objcopy can make it happen

Marc Espie

- Figured out the infrastructure part
- Can build debug packages on the sly (shadow packages)

Shadow packages

It's all a question of answering the question "What do you want ?"

- debug information is large, so it has to be opt-in
- it should be as transparent as possible, so as to not interfere with normal builds

Basic principles

- Non intrusive debug packages mean they don't actually exist
- You can't depend on them
- ... but they are still real packages

Agile process

The first version

- The normal build process does build/fake (staging area)/package
- At the end of fake and before package, extract the debug info
- During packaging, create debug packages with generated packing-list
- "opt-in" means you have to set `DEBUG_PACKAGES` to get debug packages.

multi packages

Fun logic

- OpenBSD builds once, but may create several packages at once
- Set `DEBUG_PACKAGES=${MULTI_PACKAGES}`

Do things manually first

- Also set `DEBUG_FILES` to the list of files with debug info

The code I

We replace

```
pkg_create -DPORTSDIR="${PORTSDIR}" $deps ${PKG_ARGS${_S}} $$tmp
```

with

```
${_create_pkg}
```

and we add:

```
_copy_debug_info:
.for P in ${DEBUG_FILES:N*.a}
	@dbgpath=${PREFIX}/${P:H}/.debug; \
	dbginfo=$${dbgpath}/${P:T}.dbg; \
	p=${PREFIX}/$P; \
	${INSTALL_DATA_DIR} $${dbgpath}; \
	echo "> move debug info from $$p into $${dbginfo}"; \
	objcopy --only-keep-debug $$p $${dbginfo}; \
	objcopy --strip-debug $$p; \
	objcopy --add-gnu-debuglink=$${dbginfo} $$p
```

The code II

```
.endfor
.for P in ${DEBUG_FILES:M*.a}
        @dbgpath=${PREFIX}/${P:H}/.debug; \
        dbginfo=$${dbgpath}/${P:T}; \
        p=${PREFIX}/$P; \
        ${INSTALL_DATA_DIR} $${dbgpath}; \
        echo "> copy debug info from $$p into $${dbginfo}"; \
        cp $$p $${dbginfo}; \
        strip $$p
.endfor
```

gnu debug link

the process

- gdb will look into `.debug/file.dbg` automatically
- ... provided you set `--add-gnu-debuglink=file.dbg`
- ... will error out if you keep the same name
- ... you can configure a global debug directory
- ... but having local debug directories is simple
- ... or you can use unique ids

Feedback

- During the second day, debug packages started appearing
- Size increases were "as expected"
- Setting `DEBUG_FILES` was annoying but expected
- Stripping debug info in `_copy_debug_info` is a pain in the process because you have to run `fake` all over again

V2

- We already annotate files in manifests (packing-lists) when they are binary or shared libraries
- We can reuse the same info instead of `DEBUG_FILES`
- So process all packing-lists, generate `DEBUG_FILES` from there, and the resulting packing-lists
- ... much easier to do than a year ago, because `update-plist` already processes all packing-lists at once.
- Synopsis:
  `update-plist [options] -- pkg1args pkg1name pkg2args pkg2name...`
- Create `build-debug-info` based on the same synopsis
- it creates the packing-lists, and the list of debug files

Not enough

- The process becomes more complicated
- For a new port, you must run `make fake`, `make update-plist`
- *then* you can set `DEBUG_PACKAGES` and clean the `fake` area
- because `copy_debug_info` is stateful and destructive

Getting there I

- Put more logic in build_debug_info
- Have it create a makefile, so it can be run several times

```
# Makefile generated by build-debug-info $OpenBSD: build-debug-info,v 1.27 2019/1
# No serviceable parts
# Intended to run under the stage area after cd ${WRKINST}

OBJCOPY_RULE = ${INSTALL_DATA_DIR} ${@D} && \
    echo "> Copy debug info from $? to $@" && \
    if readelf 2>/dev/null -wi $?|cmp -s /dev/null -; then \
            echo "Warning: no debug-info in $?"; \
    fi && \
    objcopy --only-keep-debug $? $@ && \
    objcopy --strip-debug $? && \
    objcopy --add-gnu-debuglink=$@ $? && \
    touch $@
```

Getting there II

```
LINK_RULE = ${INSTALL_DATA_DIR} ${@D} && \
    echo "> Link debug info from $? to $@" && ln $? $@

all:
.PHONY: all

all: bin/.debug/bsdcat.dbg
bin/.debug/bsdcat.dbg: bin/bsdcat
        @${OBJCOPY_RULE}

all: bin/.debug/bsdcpio.dbg
bin/.debug/bsdcpio.dbg: bin/bsdcpio
        @${OBJCOPY_RULE}

all: bin/.debug/bsdtar.dbg
```

Getting there III

```
bin/.debug/bsdtar.dbg: bin/bsdtar
        @${OBJCOPY_RULE}

all: lib/.debug/libarchive.so.10.3.dbg
lib/.debug/libarchive.so.10.3.dbg: lib/libarchive.so.10.3
        @${OBJCOPY_RULE}
```

When to emit debug

Build part

- having `DEBUG_PACKAGES` non empty leads to `CFLAGS += -g` and `INSTALL_STRIP` empty
- we actually set `DEBUG_PACKAGES = ${BUILD_PACKAGES}` because we don't build everything
- SUBPACKAGES with `PKG_ARCH=*` get removed
- This is a Makefile, so we can't decide in a "smart way". This may result in empty packages if there is no binary.
- In which case, you have to set `DEBUG_PACKAGES` more specifically.

So this got commited on day 4 or so

The devil lies in the details

- `objcopy` will happily extract non existent debug info
- ... hence the `readelf` check

What about links

- `gnu-link-info` is just a filename
- if you `ln bin/a libexec/b` then both names share the same info
- So `build-debug-info` records hardlinks (in the fake stage) and emits correct info
  - If another name in the same directory, nothing to do. Second name will point to the right debug file
  - If name in another directory, need to link debug files as well, so that `libexec/b` can point to `libexec/.debug/a.dbg`

Aftermath

The meson puzzle

- Ports based on `meson.port.mk` didn't work
- turns out the module was doing `.if !empty(DEBUG_PACKAGES)`
- ... but `BUILD_PACKAGES` is not yet defined at that point, it is computed after modules
- ... so `DEBUG_PACKAGES` is still empty
- Solution: preventively set `BUILD_PACKAGES` to something before modules are evaluated

Aftermath 2

The shearing issue

- Normally, you don't install debug packages
- ... but later, your snapshot gets out of synch
- ... mirrors don't keep snapshots forever.
- Solution: set `DEBUG_PKG_CACHE` so that debug packages get downloaded (and synched) automatically
- Surprisingly easy to write

The shitz I

```
sub may_grab_debug_for
{
        my ($class, $orig, $kept, $state) = @_;
        return if $orig =~ m/^debug\-/;
        my $dbg = "debug-$orig";
        return if $state->tracker->is_known($dbg);
        return if OpenBSD::PackageInfo::is_installed($dbg);
        my $d = $state->debug_cache_directory;
        return if $kept && -f "$d/$dbg.tgz";
        $class->grab_debug_package($d, $dbg, $state);
}

sub grab_debug_package
{
        my ($class, $d, $dbg, $state) = @_;
```

The shitz II

```perl
my $o = $state->locator->find($dbg);
return if !defined $o;
require OpenBSD::Temp;
my ($fh, $name) = OpenBSD::Temp::permanent_file($d, "debug-pkg");
if (!defined $fh) {
        $state->errsay(OpenBSD::Temp->last_error);
        return;
}
my $r = fork;
if (!defined $r) {
        $state->fatal("Cannot fork: #1", $!);
} elsif ($r == 0) {
        $DB::inhibit_exit = 0;
        open(STDOUT, '>&', $fh);
        open(STDERR, '>>', $o->{errors});
        $o->{repository}->grab_object($o);
```

The shitz III

```
        } else {
                close($fh);
                waitpid($r, 0);
                my $c = $?;
                $o->{repository}->parse_problems($o->{errors}, 1, $o);
                if ($c == 0) {
                        rename($name, "$d/$dbg.tgz");
                } else {
                        unlink($name);
                        $state->errsay("Grabbing debug package failed: #1",
                                $state->child_error($c));
                }
        }
}
```

thank you

Questions ?