Cross Compiling For Perl Hackers

Jens Rehsack

Niederrhein Perl Mongers

The Perl Conference in Amsterdam 2017

Motivation

The Restaurant at the End of the Universe

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.

*(*Douglas Adams*)*

Goals

Clarify some use-cases

- Cross-Compiling
- Cross-Building
- Canadian Cross
- Foreign builds

Sensibilize beyond developer environments

- How can I enable other people using my code?
- What else beside specs, tests and documentation can be provided?
- Why should I care?

Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?
- A platform is defined by
  - Architecture
  - Vendor
  - Operating System / ABI
    - i486-pc-linux-gnu
    - x86_64-apple-darwin64
    - arm926ejse-poky-linux-gnueabi
    - cortexa9hf-vfp-neon-mx6qdl-poky-linux-gnueabi
    - sparcv9-sun-solaris

API vs. ABI

size_t-size.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("%zd\n", sizeof(size_t));
    return 0;
}
```

32-bit mode size_t-size

```
$ cc -O -m32 -o size_t-size size_t-size.c
$ ./size_t-size
4
```

64-bit mode size_t-size

```
$ cc -O -m64 -o size_t-size size_t-size.c
$ ./size_t-size
8
```

API

- abbreviation for "Application Programming Interface"
- defines compatibility on source level

snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```

- every STD C conforming C program can call snprintf

snprintf invocation

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    char buf[_PATH_MAX];
    snprintf(buf, sizeof buf, "%s", argv[0]);
    return 0;
}
```

ABI

- abbreviation for "Application Binary Interface"
- defines compatibility on compiled code level

snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```

- sizes of pointers depend on memory model (segmented, flat, address width, ...)
- size of buffer size depends just on a subset of the memory model: the address width

ABI influencers

- CPU register sizes
- alignment
- packing of enums/structs
- memory model (flat vs. segmented, address width, . . .)
- calling convention (stack vs. register based, order of arguments, how many registers, . . .)
- byte order

Cross Compiling

The Hitchhiker's Guide to the Galaxy

Don't Panic.

*(*Douglas Adams*)*

Cross Compiling "Hello world"

What does such a compiler do?
Compiles source.

hello.c

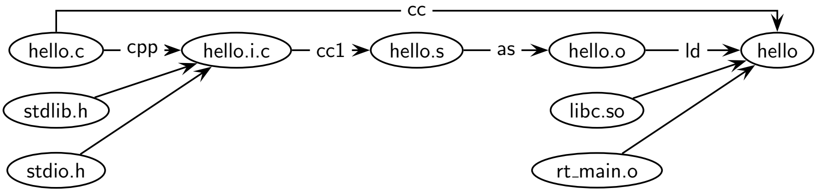
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("Hello world\n");
    return 0;
}
```

Cross Compiling II

compiler invocation

```
$ ${CC} -o hello hello.c
```



Cross Development Kit

What is this . . . you're talking about?

cpp	C PreProcessor
cc1	C Compiler
as	Assembler
ld	Linker Wrapper
collect2	Linker

Cross Development Kit II

and what does it use, too?

ar	create and maintain library archives
nm	display name list (symbol table)
objcopy	copy and translate object files
objdump	display information from object files
ranlib	generate index to archive
strings	find the printable strings in a object, or other binary, file
strip	remove symbols

Cross Development Kit Tools

```
# (cd .../arm-poky-linux-gnueabi/ && ls )
arm-poky-linux-gnueabi-addr2line
arm-poky-linux-gnueabi-ar
arm-poky-linux-gnueabi-as
arm-poky-linux-gnueabi-cpp
arm-poky-linux-gnueabi-elfedit
arm-poky-linux-gnueabi-gcc
arm-poky-linux-gnueabi-gcc-ar
arm-poky-linux-gnueabi-gcov
arm-poky-linux-gnueabi-gprof
arm-poky-linux-gnueabi-ld
arm-poky-linux-gnueabi-ld.bfd
arm-poky-linux-gnueabi-nm
arm-poky-linux-gnueabi-objcopy
arm-poky-linux-gnueabi-objdump
arm-poky-linux-gnueabi-ranlib
...
```

Cross Development Kit Location

Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdlib.h
```

Similar picture for `stdio.h`, `stdint.h`, `libc.so`, `rt_main.o`, ...

Towel

Knowing where one's towel is

Any man who can hitch the length and breadth of the galaxy, rough it, slum it, struggle against terrible odds, win through, and still know where his towel is, is clearly a man to be reckoned with.

Convinced

Where can I download it?

Which one?

Build Yourself a Cross-SDK

Use the source, Luke

There're several ways:

- the hard way: do it yourself as described at Building Cross Toolchains with gcc or Build a GCC-based cross compiler for Linux
- Toolchain build helper like crosstool-NG or Scratchbox
- Full flavoured - Yocto or T2 SDE

Vendor Cross-SDK

Typical cases

- Bare Metal SDK
- Accelerator Libraries (typically not Open-Source)
- Early Adopter
- Enterprise Support

And now

Which way I ought to go from here?

That depends . . . on where you want to get to.

Topic was . . .

Cross compiling for Perl Hackers
we didn't define an audience, reasonable possibilities are

- Perl Porters
- Perl Module Maintainers

Perl Porters probably have to care for more than Perl Module Maintainers . . .

Build here, run there

Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?
  mind HOSTCC vs. CC
- ...and which *stdlib.h*/*libc.so*?
  modern toolchains know `--sysroot` argument - prior lot's on replacements in `-I...` and `-L...` were required
- pick right CFLAGS, BUILD_CFLAGS, HOST_CFLAGS or TARGET_CFLAGS for the right job, likewise for LDFLAGS, CCLDFLAGS, LDDLFLAGS, CXXFLAGS and whatever additional tool is used
- do not mix build and target configuration
- do not run target artifacts locally

Oops ...

Host vs. Target II

```
# Failed test 1 - positive infinity at op/infnan.t line 53
#      got "0"
# expected > "0"
# 0 - 0 = 0
# Failed test 2 - negative infinity at op/infnan.t line 54
#      got "0"
# expected < "0"
```

Weird ...

Host vs. Target III

```
Anibal Limon <anibal.limon@intel.com>
> Also do you know why we are using config.sh file instead of leave perl
> generate it on do\_configure?

Perl's configure stage doesn't know neither cares for cross compiling.
It would determine a lot of host related configuration instead of target
related one.

It might be reasonable to collect and analyze output (config.sh) of
./Configure for a lot of different machines (eg. Cortex-A9 vs. XScale,
vs. Kirkwood vs. ..., you see, I had to much ARM, to less PPC or MIPS
...) to derive much better puzzled config.sh ...
```

Build here, run there II

Build vs. Run

- during build, several development kits are involved (at least host and target, sometimes host, build and multiple targets)
- PATHs vary, eg. `-L/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/lib` vs. `-Wl,-R/usr/lib`

Build here, run there III

mind those differences when invoking wrapper-scripts

Build vs. Run

```
rakudo-star % make install
# This is a post-compile task, unfortunately placed into install stage
./perl6-j tools/build/install-core-dist.pl /foo-bsp/w/tmp/work/...
    cortexa9hf-vfp-neon-poky-linux-gnueabi/rakudo-star/2016.01-r0/...
    image/usr/share/nqp
Error: Could not find or load main class perl6
```

perl6-j

```
#!/bin/sh
: ${NQP_DIR:="/usr/share/nqp"}
: ${NQP_JARS:="${NQP_DIR}/runtime/asm-4.1.jar:${NQP_DIR}/runtime/asm-tree-4.1.jar:\
               ${NQP_DIR}/runtime/jline-1.0.jar:${NQP_DIR}/runtime/jna.jar:\
               ${NQP_DIR}/runtime/nqp-runtime.jar:${NQP_DIR}/lib/nqp.jar"}
: ${PERL6_DIR:="/usr/share/perl6"}
: ${PERL6_JARS:="${NQP_JARS}:${PERL6_DIR}/runtime/rakudo-runtime.jar:${PERL6_DIR}/runtime/perl6.jar"}
exec java -noverify -Xms100m -Xbootclasspath/a:${NQP_JARS}:${PERL6_DIR}/runtime/rakudo-runtime.jar:\
        ${PERL6_DIR}/runtime/perl6.jar -cp $CLASSPATH:${PERL6_DIR}/runtime:${PERL6_DIR}/lib:\
        ${NQP_DIR}/lib -Dperl6.prefix=/usr -Djna.library.path=/usr/share/perl6/site/lib \
        -Dperl6.execname="$0" perl6 "$@"
```

Build here, run there IV

Build vs. Run

- guess why running that script from `/foo-bsp/w/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/...` `rakudo-star/2016.01-r0/rakudo-star-2016.01/` fails ...
- remember `sdkroot` (build libraries, can be executed in build environment) and `sysroot` (target runtime chroot, used for linking etc.)
- all path's in `sysroot` are as if the files were already on `target`

Experience

The Salmon of Doubt

A learning experience is one of those things that says, "You know that thing you just did? Don't do that."

(Douglas Adams)

Configure Stage

Prerequisites . . .

- nowadays Perl Toolchain doesn't support cross compile dependcy checks
- neither external resources (mind wrapper modules as RRDTool::OO), so configure stage has to prove on it's own (compile and link test in *Makefile.PL*)
- `x_prereqs` was an idea but never completed
- workaround in Yocto for module prerequisites: DEPENDS (configure stage) contain host packages, RDEPENDS (install stage) contain target packages
  - ☞ it's slightly more complicated for external libraries when *Makefile.PL* doesn't know about cross compiling

Questions

What can we prove?

- Everything which can be figured out without
  1. executing
  2. accessing target
- That's it

Proving Sizes

Easy way

```
use Config::AutoConf;
Config::AutoConf->check_sizeof_type( "ASN1_CTX", {
    prologue => "#include <openssl/ssl.h>" } );
```

- load neat helper
- ask for proving size of ASN1_CTX
- load includes defining the proved type

Behind the Scenes I

Generated code

```
/* end of conftest.h */
#include <openssl/ssl.h>

int
main ()
{
static int test_array [(((long int)(sizeof(ASN1_CTX))) <= %SIZE%) ? 1 : -1];
test_array [0] = 0 ;
return 0;
}
```

- We asked to include "ssl.h" without approval (test scene)
- Sizes are tested using arrays with negative size on test failure
- need to assign to array to avoid optimizing it away
- return to avoid compiler error as false negative

☞ run that code though C compiler using a binary search algorithm for %SIZE%

Searching for size

Sample run

```
/* end of conftest.h */

#include <openssl/ssl.h>

  int
  main ()
  {
    static int test_array [(((long int)(sizeof(ASN1_CTX))) <= 2) ? 1 : -1 ];
  test_array [0] = 0
;
    return 0;
  }
```

Sample failure

```
testf8cYAi.c:8:30: error: 'test_array' declared as an array
with a negative size
```

A mini attempt on "w"

Requirements Analysis

- get list of logged in users
- get list processes with command and uid
- get general information (uptime, load, . . .)

⇒ For example let's focus on logged in users

Prepare probes for "w"

```
use Config::AutoConf;
my $ac = Config::AutoConf->new();
$ac->check_prog_cc; $ac->check_default_headers;
if($ac->check_headers("utmpx.h", "utmp.h")) {
  my $utx_incl = $ac->_default_includes . q[
#ifdef HAVE_UTMP_H
#include <utmp.h>
#endif
#ifdef HAVE_UTMPX_H
#include <utmpx.h>
#endif
];
  ...
```

- get an own instance of `Config::AutoConf`
- prove basics(compiler, reasonable headers)
- prove from most modern to least modern whether utmp support is available
- extend default prologue for further testing

Probe listing logged in users "w"

```
  ...
  if($ac->check_type("struct utmpx", {prologue => $utx_incl})) {
    $ac->check_members( ["struct utmpx.ut_type", "struct utmpx.ut_host"], {
      prologue => $utx_incl });
    $ac->check_funcs([qw[getutxent setutxent endutxent]], {prologue => ...});
    $ac->check_decls([qw[getutxent setutxent endutxent]], {prologue => ...});
    ...
  }
  elsif($ac->check_type( "struct utmp", {prologue => $utx_incl})) {
    ...
    $ac->check_file( "/var/tmp/utmp" )
      and $ac->define_var( "UTMP_DB", "/var/tmp/utmp" );
  }
}
$ac->write_config_h
```

- check whether `struct utmpx` is available (might be through `utmp.h`)
- prove which data (`members`) are available through `struct utmpx`
- Check the availability of functions (`getutxent`, `setutxent`, `endutxent`, ...) and whether they have a declaration
- Same procedure on `struct utmp` when `struct utmpx` wasn't found
- When all else fails check well known db locations ...

Anatomy of proves

Built the fundament

1. check for required tools
2. check headers
3. check types
4. check functions
5. check declarations

This order is valid for global (`stdint.h`) and local (`utmpx.h`) probes.

Conclusion

- stay as close as possible to existing standards - reinventing the wheel will almost always fail
- use ExtUtils::MakeMaker for building
- use Config::AutoConf when it is really necessary to have configure time checks (as which API is supported by wrapped library)
- prefer pkgconf (or pkg-config) over compile and link testing
- always allow every check being overwritten by environment variables

Resources

Cross Compile Perl

[P5P] Remodeling the cross-compilation model
http://grokbase.com/t/perl/perl5-porters/141gz52519/remo

Cross Compile Guides

Building Cross Toolchains with gcc
https://gcc.gnu.org/wiki/Building_Cross_Toolchains_with_

Build a GCC-based cross compiler for Linux
https://www6.software.ibm.com/developerworks/education/l

Resources

Cross Compile Helper

crosstool-NG	http://crosstool-ng.org/
Scratchbox	http://www.scratchbox.org/

Cross Compile Distribution Builder

Yocto	http://yoctoproject.org/
T2 SDE	http://t2-project.org/

Thank You For Listening

Questions?

Jens Rehsack <rehsack@cpan.org>
Cologne