

MK-COFIGURE – lightweight easy for use replacement for GNU Autotools

Aleksey Cheusov
v1e@gmx.net

LVEE 2009

What's wrong with GNU autotools?

- ▶ Too complex for use
- ▶ Overbloated **** (tens of thousands(!) lines of source code)
- ▶ Inconvenient for development, too many tools and files to deal with
- ▶ Generated configure/Makefile are too big (hundreds of kilobytes for trivial configure.ac)
- ▶ Generated configure/Makefile are too slow even on modern hardware
- ▶ No results caching
- ▶ Many (most?) real life autotools-based applications don't support/break cross-building
- ▶ In my view, autotools are extremely badly designed and implemented :-P

What's wrong conceptually with GNU autoconf?

What's bad

- ▶ Code generation approach
- ▶ Orientation on POSIX make
- ▶ m4 for autoconf
- ▶ Too many tools and files (`autoconf`, `autoheader`, `automake`, `autoreconf`, `Makefile`, `Makefile.in`, `Makefile.am`, `configure.ac`, `config.guess`, `config.sub`, `config.status`, `configure`, `aclocal`, `config.h.in`, `config.h`, `install-sh`, `aclocal.m4`, `make`).

What's good

- ▶ End-users and packagers need not install any external software to build autotools-based applications.

Concepts behind mk-configure

bla-bla-bla :-)

Design principles and goals

- ▶ Declarative approach in writing Makefiles
- ▶ Keep It Short and Simple
- ▶ No code generation. Library approach is used instead.
- ▶ No POSIX make. It is too limited.
- ▶ “NetBSD bmake is magic-enough” ©. The only command you should know about is bmake.
- ▶ Everything is in **Makefile**.

Implementation

- ▶ mk-configure uses `bsd.*.mk` files (AKA mk-files)

Bad effect

- ▶ End-users/packagers have to install bmake and mk-configure to build mk-configure-based applications.

Example 1: hello world

Makefile

```
PROG=      hello
MKMAN=     no
WARNS=     4

.include <mkc.prog.mk>
```

hello.c

```
#include <stdio.h>

int main (int, char **)
{
    puts ("Hello World!");
    return 0;
}
```

Example 1: hello world

everybody loves screenshots :-)

shell session

```
$ bmake
gcc -Wno-error -Wall -Wstrict-prototypes
  -Wmissing-prototypes -Wpointer-arith
  -Wno-uninitialized -Wreturn-type -Wcast-qual
  -Wpointer-arith -Wwrite-strings -Wswitch
  -Wshadow -Werror -c hello.c
gcc -o hello hello.o
$ ./hello
Hello World!
$ bmake PREFIX=/tmp/temp/destdir install-dirs
install -d /tmp/temp/destdir/bin
$ bmake PREFIX=/tmp/temp/destdir install
install -c -s -o 10040 -g 10001 -m 555
  hello /tmp/temp/destdir/bin/hello
$
```

Example 2: using non-standard function strncpy(3)

Makefile

```
MKC_SOURCE_FUNCLIBS=  strncpy
MKC_CHECK_FUNCS3=    strncpy:string.h

PROG=                 strncpy_test
SRCS=                 main.c

.include <mkc.configure.mk>
.include <mkc.prog.mk>
```

Example 2: using non-standard function strncpy(3)

strncpy_test.c

```
#include <string.h>

#ifdef HAVE_FUNC3_STRLCPY_STRING_H
size_t strncpy(char *dst, const char *src, size_t siz);
#endif

int main ()
{
    /*    Use strncpy(3) here    */
    return 0;
}
```

Example 2: using non-standard function strlcpy(3)

Linux

shell session

```
$ bmake
checking for function strlcpy... no
checking for func strlcpy ( string.h )... no
gcc -Wno-error -Werror -c main.c
gcc -Wno-error -Werror -c strlcpy.c
gcc -o strlcpy_test main.o strlcpy.o
$ ls -a _mkc_*
_mkc_func3_strlcpy_string_h.c
_mkc_func3_strlcpy_string_h.err
_mkc_func3_strlcpy_string_h.res
_mkc_funclibs_strlcpy.c
_mkc_funclibs_strlcpy.err
_mkc_funclibs_strlcpy.res
$
```

Example 2: using non-standard function strlcpy(3)

NetBSD

shell session

```
$ make
checking for function strlcpy... yes
checking for func strlcpy ( string.h )... yes
# compile main.o
cc -O2 -DHAVE_FUNC3_STRLCPY_STRING_H=1 -Werror -c main.c
# link strlcpy_test
cc -o strlcpy_test main.o -Wl,-rpath-link,/lib -L/lib
-Wl,-rpath-link,/usr/lib -L/usr/lib
$
```

Example 3: optional plugin support

do not sleep! :-)

Makefile

```
MKC_CHECK_FUNCLIBS=    dlopen:dl

PROG=                  progname

.include <mkc.configure.mk>

.if ${HAVE_FUNCLIB.dlopen} || ${HAVE_FUNCLIB.dlopen.dl}
CFLAGS+= -DPLUGINS_ENABLED=1
.endif

.include <mkc.prog.mk>
```

Example 4: gettimeofday(2) or obsolete ftime(3)

Makefile

```
MKC_CHECK_FUNCLIBS=    gettimeofday ftime:compat
MKC_NOAUTO_FUNCLIBS=  ftime:compat
LIB=                   mega-lib
SHLIB_MAJOR=           0
SHLIB_MINOR=           0

.include <mkc.configure.mk>

.if ${HAVE_FUNCLIB.gettimeofday}
CFLAGS+= -DUSE_GETTIMEOFDAY
.elif ${HAVE_FUNCLIB.ftime}
CFLAGS+= -DUSE_FTIME
.elif ${HAVE_FUNCLIB.ftime.compat}
CFLAGS+= -DUSE_FTIME
LDADD+= -lcompat
.endif

.include <mkc.lib.mk>
```

MK-CONFIGURE features

Module `mk.configure.mk` provides:

1. Checks for
 - ▶ header presence (`MKC_CHECK_HEADERS`)
 - ▶ function declaration (`MKC_CHECK_FUNCS[n]`)
 - ▶ type declaration (`MKC_CHECK_TYPES`)
 - ▶ structure member (`MKC_CHECK_MEMBERS`)
 - ▶ variable declaration (`MKC_CHECK_VARS`)
 - ▶ define declaration (`MKC_CHECK_DEFINES`)
 - ▶ type size (`MKC_CHECK_SIZEOF`)
 - ▶ function implementation in the library (`MKC_CHECK_FUNCLIBS` and `MKC_SOURCE_FUNCLIBS`)
2. common defines for all checks and building (`MKC_COMMON_DEFINES` and `MKC_COMMON_DEFINES.[system]`)
3. common header files for all checks (`MKC_COMMON_HEADERS`)

MK-CONFIGURE features

Module `mkc.configure.mk` provides:

4. check results passed to compiler through defines (e.g. `-DHAVE_HEADER_STRINGS_H=1`)
5. Makefile-level variables (e.g. `HAVE_HEADER.strings_h=1`)
6. other features

MK-CONFIGURE features

Module [mkc.prog.mk](#) provides:

- ▶ support for building and installing executables written in and/or with a help of C, C++, Fortran, Yacc, Lex etc.

Module [mkc.lib.mk](#) provides:

- ▶ building and installing static and shared libraries

Module [mkc.man.mk](#) provides:

- ▶ building and installing .man/.cat pages

Module [mkc.info.mk](#) provides:

- ▶ building and installing .texinfo documentation

Module [mkc.files.mk](#) provides:

- ▶ installing the text files e.g. scripts, documentation files etc.

Module [mkc.subdir.mk](#) provides:

- ▶ support for multi-directory big projects

MK-CONFIGURE features

Module `mkc.suggestions.mk` is welcome ;-)

Module `mkc.intexts.mk` provides:

- ▶ substituting `@syscondfir@`, `@datadir@` etc.

Standalone shell scripts for checking.

- ▶ `mkc_check_header` - header files.
- ▶ `mkc_check_sizeof` - `sizeof (type)`
- ▶ `mkc_check_decl` - types, variables, functions, defines and struct members
- ▶ `mkc_check_funclib` - function in library

Missed features, Plans and TODO

- ▶ mk-configure uses `bsd.*.mk` files which are unfortunately gcc-oriented. Support for other compilers is needed. Support for libtool is needed too.
- ▶ Support for custom checks.
- ▶ Shell-based equivalent.

mkc_config

```
MKC_CONFIG_MK=mkc_config.mk
MKC_CONFIG_H=mkc_config.h

MKC_CHECK_HEADERS='
  strings.h malloc.h poll.h'
MKC_CHECK_FUNCLIBS='
  dlopen:dl memalign socket:socket'

. mkc_configure.sh
mkc_checks
mkc_gen # generating mkc_config.{h,mk}
```

how to build

```
sh mkc_config
make
make install
```

PR

```
Web Site
  Community
  Tutorial
  slashdot/papers
  Wiki
  popularization
  packages
```