Building static executables on Linux

Anatol Belski

Friday, February 29. 2008

Building static executables on Linux

An interesting question I had to answer a couple months ago was - how could I compile an executable statically, so then it works on various linux distributions without the need to install some additional libs. Of course, such the executables are bigger as that, which was compiled in order to use the dynamic libs. But if one wanna to have an independent production executable, this is what one could need.

So, I will explain the thing with the help of an example program, which uses ncurses. First I'm creating the project dir, for example:

BASH:

```
user@host~$ mkdir ~/programming/static_test
user@host~$ cd ~/programming/static_test
```

Now I'll create the whole folder structure of the project:

BASH:

```
user@host~$ mkdir inc
user@host~$ mkdir lib
user@host~$ mkdir obj
user@host~$ mkdir out
user@host~$ mkdir src
```

So, let me explain, why I'm doing the scheme like this:

- the lib dir will contain all the *.a files, needed to compile statically
- the inc dir will contain all the headers to the lib/*.a files
- the obj dir, where the compiled *.o files will be placed in
- the src dir, where our source files are placed in
- the out dir, where all the compiled binaries etc. are placed in

So now, get the ncurses package, for example from here: ftp://ftp.gnu.org/gnu/ncurses/. Note, you can surely simply install the ncurses-dev package from your favorite distro, eq. like "apt-get install libncurses-dev" on debian, but than you'll get no example file we need. In the original gnu distribution the demo file can be located in the c++ dir, so I'm doing something like this, to use it:

BASH:

```
user@host~$ cp ~/programming/ncurses-5.6/c++/demo.cc ~/static_test/demo.cpp
```

So, we are using .cpp extentions for c++ sources under linux.

The next step would be to compile ncurses:

BASH:

```
user@host~$ cd /programming/ncurses-5.6
user@host~$ ./configure && make
```

and to copy all the archive file *.a to our lib dir:

BASH:

```
user@host~$ cd ~/programming/ncurses-5.6/lib
user@host~$ cp libform.a libmenu.a libncurses.a libncurses++.a libpanel.a ~/programming/static_test/lib
```

But this are not really all the libs we need to create a fully statically binary. Also, we need:

BASH:

```
user@host~$ cp `g++ -print-file-name=libstdc++.a` ~/programming/satic_test/lib/libstdc++.a
user@host~$ cp `g++ -print-file-name=libm.a` ~/programming/static_test/lib/libm.a
user@host~$ cp `g++ -print-file-name=libc.a` ~/programming/static_test/lib/libc.a
```

If this .a files aren't on your system ... simply install the appropriate development packages (you don't need to compile them from the scratch).

The next and last thing we need to make our project all-sufficient are the headers:

BASH:

```
user@host~$ cd ~/programming/ncurses-5.6
```

```
user@host~$ cd ~/programming/ncurses-5.6
user@host~$ cp c++/cursesapp.h c++/cursesf.h c++/curses.h c++/cursesm.h c++/cursesp.h \
c++/cursesw.h c++/cursslk.h form/form.h menu/menu.h panel/panel.h include/nc*.h include/tic.h  \
include/unctrl.h ~/programming/static_test/inc
```

hm ... it looks to be complete *.h list ... but if something isn't present, look for it in the ncurses src dir or in the /usr/include ... I've done it a couple of months ago, so it may be not quite equal for you.

The thing one needs to make a project beautifull is a make or a configure script ... so, if some specific project is being maked, which must be compiled on the multiple platforms, you need the ./configure script (and hence automake etc.). If you're using it only to compile some project without this, it's enough to make a make script (as i'm doing it). It's not really bash, but at the moment I've no plugin to make it visible ..: so it's my Makefile (note, it's not really bash, it's make):

BASH:

```
SRC_DIR = ./src
OUT_DIR = ./out
OBJ_DIR = ./obj
INC_DIR = ./inc
LIB_DIR = ./lib

INCLUDE = -I$(INC_DIR)
LIB = -L$(LIB_DIR) -lncurses++ -lform -lmenu -lpanel -lncurses

VPATH = $(INC_DIR):$(OBJ_DIR):$(SRC_DIR):$(LIB_DIR)
CXXFLAGS = $(INCLUDE) -O2

demo:   demo.o\
        libncurses++.a  \
        libform.a libmenu.a libpanel.a libncurses.a \
        libstdc++.a libm.a
        g++ -static-libgcc -o $(OUT_DIR)/demo $(OBJ_DIR)/demo.o $(LIB)

demo.o: demo.cpp internal.h ncurses_cfg.h ncurses_def.h \
        curses.h unctrl.h cursesm.h cursesp.h panel.h \
        menu.h cursesf.h form.h
        g++ -o $(OBJ_DIR)/demo.o -DNDEBUG $(CXXFLAGS) -c $(SRC_DIR)/demo.cpp

clean:
        rm -f $(OBJ_DIR)/*
        rm -f $(OUT_DIR)/*
```

Notice the -static-libgcc g++ param, without which it'll not compile statically (because there is no statically compiled libgcc.a or something else like this).

Lern more about the make files there http://www.gnu.org/software/make/manual/ or simply by the printed one.

so then, the following will make the whole project:

BASH:

```
user@host~$ cd ~/programming/static_test
user@host~/programming/static_test$ make
```

Now we got a binary in our out dir ... it's being compiled statically .. it can be used on each platform, your platform is related for. So in my case, the binary works on every 32 bit linux distribution I'm trying.

BASH:

```
user@host~$ ldd out/demo
        not a dynamic executable
user@host~$ file out/demo
out/demo: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.4.1, statically linked, for GNU/Linux 2.4.1, not stripped
```

Posted by Anatol Belski in C/C++, Linux at 19:05 | Comments (0) | Trackbacks (0) TWEET THIS

Trackbacks

Trackback specific URI for this entry

No Trackbacks

Comments

Display comments as (Linear | Threaded)

No comments

Add Comment

Name

Email

Homepage

In reply to [Top level]

Comment

Standard emoticons like :-) and ;-) are converted to images.
E-Mail addresses will not be displayed and will only be used for E-Mail notifications.
BBCode format allowed
BBCode format allowed

Remember Information?
Subscribe to this entry

Submitted comments will be subject to moderation before being displayed.

Submit Comment Preview