

Introduction of Makefile in Minix

Note: this document is fully tested only on Minix3.1.2a.

In this document, we look into some sample Makefiles in Minix.

'*Make*' is an powerful tool, for official manual, please refer to:

<http://www.gnu.org/software/make/manual/>

For brief introduction, please refer to:

<http://frank.mtsu.edu/~csdept/FacilitiesAndResources/make.htm>

However, what we'll talk in this document should be enough for you to survive the Labs.

I. Parse the Makefile

Make is a GNU tool for managing large project. What the tool does is to parse the Makefile and execute operations exactly following the rules in Makefile.

Here, we Parse the `/usr/src/servers/inet/Makefile`, note '#' starts comment till the end of the line. To make it clear, I use '/' as comment, just for explanation. *Makefile* do NOT take '/' as comment.

```
# Makefile for inet.
```

```
# Directories
```

```
g = generic // this is a macro statement.
```

```
# Programs, flags, and libraries
```

```
CC = cc // indicate which compiler we are using, you can use gcc if
// it is installed in your minix.
```

```
CPPFLAGS = -I. -D_MINIX // flags needs to be provided when compiling
```

```
CFLAGS = $(OPT) $(CPPFLAGS) // $(XXX) is how we use the macro in
// statement.
// OPT is internally defined macros which
// are ones that are predefined in make, it
// is null here.
```

```
LDFLAGS =
```

```
LIBS = -lsys -lsysutil //declare the libraries we are gonna use
```

```
.c.o: // suffix or implicit rule, which tells make
// that .o object files are made from .c
// source files. Or from make's point of view, the target .o files
// depends on .c files. We'll see more examples soon.
$(CC) $(CFLAGS) -o $@ -c $< // tells make how to make .o object from
// .c files. We have two special macro here:
// $@ equals target .o file.
// $< equals .c file
```

```

OBJ = buf.o clock.o inet.o inet_config.o \           //all the object files we need to produce
      mnx_eth.o mq.o qp.o sr.o stacktrace.o \
      $g/udp.o $g/arp.o $g/eth.o $g/event.o \
      $g/icmp.o $g/io.o $g/ip.o $g/ip_ioctl.o \
      $g/ip_lib.o $g/ip_read.o $g/ip_write.o \
      $g/ipr.o $g/rand256.o $g/tcp.o $g/tcp_lib.o \
      $g/tcp_recv.o $g/tcp_send.o $g/ip_eth.o \
      $g/ip_ps.o $g/psip.o \
      minix3/queryparam.o sha2.o                    // last line doesn't have '\` at end.

all:  inet           // this is the first explicit dependency rule, so make will first find
                    // this entry if no argument is provided, which means issuing 'make'
                    // or 'make all' makes no difference.
                    // And this dependency rule tells make that all is depend on inet.
                    // And make will keep looking for rule of making inet.
                    // Note there is no any operation in the rule

inet:  $(OBJ)        // rule of making inet. We can tell inet depends on all the object file
      $(CC) -o $@ $(LDFLAGS) $(OBJ) version.c $(LIBS) //the operation required
                                                    // to make inet.

install: inet
      install -c $? /usr/sbin/inet // this operation puts the binary file inet to /usr/sbin/

clean: // it doesn't depends on any thing, so when you issue 'make clean' it simply
      // do the operation below
      rm -f $(OBJ) inet *.bak

depend:
      /usr/bin/mkdep "$(CC) -E $(CPPFLAGS)" *.c generic/*.c > .depend

# Include generated dependencies.
include .depend // same as '#include' macro in C language, simply put all the content
                // of .depend file here.
                // Then by looking into the .depend file, you probably get to know
                // that one major advantage of the dependency rule is to check if
                // the target which depends on something needs to be remake or not.
                // For example, if we modify a header file, make will only remake
                // the .o files which depend on the header file we modified. For a
                // large project, it can save quite amount of time.

#
# $PchId: Makefile.mnx3,v 1.1 2005/06/28 14:28:45 philip Exp $
#

```

II. Special case in Minix3.1.2a

In Minix3.1.2a, some *Makefile* is complicated, and I don't suggest you to modify that kind of mass, like */usr/src/lib/Makefile*.

The fact is, all the *Makefiles* under */usr/src/lib* and its subdirectory are generated from *Makefile.in* by a shell script */usr/src/lib/generate.sh*. So what we need to do is to modify or create *Makefile.in* and use shell script to generate the *Makefile* for us.

But how we use that? Look into the */usr/src/lib/Makefile*, the first few lines tell you that all you need to do after modifying *Makefile.in* is simply to issue 'make'.

III. Conclusion

It is not that hard at all, isn't it? Well, actually, *make* is not necessarily used to do compiling related work. Take advantage of the mechanism, and you'll find it useful in many respects.