//首先声明在使用本文档的时候,用的命令是 make all,也就是说只执行all命令

//再次进行操作的时候需要先使用make clean 将先前创建的一些文件删除,以免产生冲突

# if you want the ram-disk device, define this to be the

# size in blocks.

#

//#是注释

//之后为定义变量,取用各个变量的时候要用$符号

RAMDISK = #-DRAMDISK=512

AS86 =as86 -0 -a

LD86 =ld86 -0

AS =as

LD =ld

LDFLAGS =-m elf\_i386 -Ttext 0 -e startup\_32

CC =gcc-3.4 -march=i386 $(RAMDISK)

CFLAGS =-m32 -g -Wall -O2 -fomit-frame-pointer

CPP =cpp -nostdinc -Iinclude

#

# ROOT\_DEV specifies the default root-device when making the image.

# This can be either FLOPPY, /dev/xxxx or empty, in which case the

# default of /dev/hd6 is used by 'build'.

#

ROOT\_DEV= #FLOPPY

//定义了后面的文件所需要依赖的文件(以路径的形式给出)

ARCHIVES=kernel/kernel.o mm/mm.o fs/fs.o

DRIVERS =kernel/blk\_drv/blk\_drv.a kernel/chr\_drv/chr\_drv.a

MATH =kernel/math/math.a

LIBS =lib/lib.a

.c.s:

$(CC) $(CFLAGS) \

-nostdinc -Iinclude -S -o $\*.s $<

.s.o:

$(AS) -o $\*.o $<

.c.o:

$(CC) $(CFLAGS) \

-nostdinc -Iinclude -c -o $\*.o $<

all: Image

//all依赖于image,即我们使用all命令的时候,image会执行,image依赖的文件也会执行

//所以image就创建出来了

//使用规则:

Target:prerequisites

(tab键)command //tab键指示这个是命令行,也就可以理解为shell模式

//Target:目标文件,即执行完了之后被(显式或隐式)创建出来

//prerequisites:依赖条件,有一些隐式规则下的文件有可能不被列出来.

如:N.o文件依赖于N.c文件,那么N.c文件可以不列出来,命令也可以使用默认命令

//command:命令行,具体要执行的命令

Image: boot/bootsect boot/setup tools/system tools/build //image依赖的文件1.bootsect

cp -f tools/system system.tmp 2.setup

strip system.tmp 3.system

objcopy -O binary -R .note -R .comment system.tmp tools/kernel 4.build

tools/build boot/bootsect boot/setup tools/kernel $(ROOT\_DEV) > Image

rm system.tmp

rm tools/kernel -f

sync

disk: Image

dd bs=8192 if=Image of=/dev/fd0

BootImage: boot/bootsect boot/setup tools/build

tools/build boot/bootsect boot/setup none $(ROOT\_DEV) > Image

sync

tools/build: tools/build.c

gcc $(CFLAGS) \

-o tools/build tools/build.c

boot/head.o: boot/head.s

gcc-3.4 -m32 -g -I./include -traditional -c boot/head.s

//当在执行makefile时,遇见include会暂停makefile的工作,转而执行include

mv head.o boot/

//找到路径下相应的文件,将它们链接成为system

tools/system: boot/head.o init/main.o \ // ‘\’表示续行符

$(ARCHIVES) $(DRIVERS) $(MATH) $(LIBS)

$(LD) $(LDFLAGS) boot/head.o init/main.o \

$(ARCHIVES) \

$(DRIVERS) \

$(MATH) \

$(LIBS) \

-o tools/system

nm tools/system | grep -v '\(compiled\)\|\(\.o$$\)\|\( [aU] \)\|\(\.\.ng$$\)\|\(LASH[RL]DI\)'| sort > System.map

kernel/math/math.a: FORCE

//第一次出现FORCE,表示无论如何总认为这个依赖被更新过,从而这个规则能一直被执行

//原因:当依赖文件的更新时间戳晚于目标文件时(即依赖文件被更新),那么该目标文件规则

//将被重新执行,目标文件将被重建

(cd kernel/math; make)

//该条命令表示进入这个文件夹内,执行文件夹内的Makefile文件

//make命令在默认情况下会寻找当前目录下的makefile文件,然后执行makefile中的

//第一个规则

//以下同

kernel/blk\_drv/blk\_drv.a: FORCE

(cd kernel/blk\_drv; make)

//命令行“**cd**”改变目录不会对其后的命令的执行产生影响。

//就是说其后的命令执行的工作目录不会是之前使用“**cd**”进入的那个目录

//所以cd需要和后面使用的命令一起

kernel/chr\_drv/chr\_drv.a: FORCE

(cd kernel/chr\_drv; make)

kernel/kernel.o: FORCE

(cd kernel; make)

mm/mm.o: FORCE

(cd mm; make)

fs/fs.o: FORCE

(cd fs; make)

lib/lib.a: FORCE

(cd lib; make)

//由于汇编语言中不能将N.s文件一步到位地变为N文件,所以要使用两条命令

//将setup.s转为setup 以下同

boot/setup: boot/setup.s

$(AS86) -o boot/setup.o boot/setup.s

//取变量AS86的值,执行命令

$(LD86) -s -o boot/setup boot/setup.o

boot/bootsect: boot/bootsect.s

$(AS86) -o boot/bootsect.o boot/bootsect.s

$(LD86) -s -o boot/bootsect boot/bootsect.o

tmp.s: boot/bootsect.s tools/system

(echo -n "SYSSIZE = (";ls -l tools/system | grep system \

| cut -c25-31 | tr '\012' ' '; echo "+ 15 ) / 16") > tmp.s

cat boot/bootsect.s >> tmp.s

//clean的作用是将前面make时创建的文件删除,以免再次make的时候产生冲突

clean:

rm -f Image System.map tmp\_make core boot/bootsect boot/setup

//rm表示删除

rm -f init/\*.o tools/system tools/build boot/\*.o

(cd mm;make clean)

(cd fs;make clean)

(cd kernel;make clean)

(cd lib;make clean)

backup: clean

(cd .. ; tar cf - linux | compress16 - > backup.Z)

sync

dep:

sed '/\#\#\# Dependencies/q' < Makefile > tmp\_make

(for i in init/\*.c;do echo -n "init/";$(CPP) -M $$i;done) >> tmp\_make

cp tmp\_make Makefile

(cd fs; make dep)

(cd kernel; make dep)

(cd mm; make dep)

# Force make run into subdirectories even no changes on source

FORCE:

### Dependencies:

init/main.o: init/main.c include/unistd.h include/sys/stat.h \

include/sys/types.h include/sys/times.h include/sys/utsname.h \

include/utime.h include/time.h include/linux/tty.h include/termios.h \

include/linux/sched.h include/linux/head.h include/linux/fs.h \

include/linux/mm.h include/signal.h include/asm/system.h \

include/asm/io.h include/stddef.h include/stdarg.h include/fcntl.h