程序 5-1 linux/Makefile 文件

```
1  #
2  # if you want the ram-disk device, define this to be the
3  # size in blocks.
4  #
   # 如果你要使用 RAM 盘(RAMDISK)设备的话就定义块的大小。这里默认 RAMDISK 没有定义（注释掉了），
   # 否则 gcc 编译时会带有选项'-DRAMDISK=512'，参见第 13 行。
5  RAMDISK = #-DRAMDISK=512
6
7  AS86    =as86 -0 -a         # 8086 汇编编译器和连接器，见列表后的介绍。后带的参数含义分别
8  LD86    =ld86 -0            # 是：-0 生成 8086 目标程序；-a 生成与 gas 和 gld 部分兼容的代码。
9
10 AS      =gas                # GNU 汇编编译器和连接器，见列表后的介绍。
11 LD      =gld
   # 下面是 GNU 链接器 gld 运行时用到的选项。含义是：-s 输出文件中省略所有的符号信息；-x 删除
   # 所有局部符号；-M 表示需要在标准输出设备(显示器)上打印连接映像(link map)，是指由连接程序
   # 产生的一种内存地址映像，其中列出了程序段装入到内存中的位置信息。具体来讲有如下信息：
   # • 目标文件及符号信息映射到内存中的位置；
   # • 公共符号如何放置；
   # • 连接中包含的所有文件成员及其引用的符号。
12 LDFLAGS =-s -x -M

   # gcc 是 GNU C 程序编译器。对于 UNIX 类的脚本(script)程序而言，在引用定义的标识符时，需在前
   # 面加上$符号并用括号括住标识符。
13 CC      =gcc $(RAMDISK)

   # 下面指定 gcc 使用的选项。前一行最后的'\'符号表示下一行是续行。选项含义为：-Wall 打印所有
   # 警告信息；-O 对代码进行优化。'-f 标志'指定与机器无关的编译标志。其中-fstrength-reduce 用
   # 于优化循环语句；-fcombine-regs 用于指明编译器在组合编译阶段把复制一个寄存器到另一个寄存
   # 器的指令组合在一起。-fomit-frame-pointer 指明对于无需帧指针（Frame pointer）的函数不要
   # 把帧指针保留在寄存器中。这样在函数中可以避免对帧指针的操作和维护。-mstring-insns 是
   # Linus 在学习 gcc 编译器时为 gcc 增加的选项，用于 gcc-1.40 在复制结构等操作时使用 386 CPU 的
   # 字符串指令，可以去掉。
14 CFLAGS  =-Wall -O -fstrength-reduce -fomit-frame-pointer \
15 -fcombine-regs -mstring-insns

   # 下面 cpp 是 gcc 的前(预)处理器程序。前处理器用于进行程序中的宏替换处理、条件编译处理以及
   # 包含进指定文件的内容，即把使用'#include'指定的文件包含进来。源程序文件中所有以符号'#'
   # 开始的行均需要由前处理器进行处理。程序中所有'#define'定义的宏都会使用其定义部分替换掉。
   # 程序中所有'#if'、'#ifdef'、'#ifndef'和'#endif'等条件判别行用于确定是否包含其指定范围中
   # 的语句。
   # '-nostdinc -Iinclude'含义是不要搜索标准头文件目录中的文件，即不用系统/usr/include/目录
   # 下的头文件，而是使用'-I'选项指定的目录或者是在当前目录里搜索头文件。
16 CPP     =cpp -nostdinc -Iinclude
17
18 #
19 # ROOT_DEV specifies the default root-device when making the image.
20 # This can be either FLOPPY, /dev/xxxx or empty, in which case the
21 # default of /dev/hd6 is used by 'build'.
22 #
   # ROOT_DEV 指定在创建内核映像(image)文件时所使用的默认根文件系统所
```

```
# 在的设备，这可以是软盘(FLOPPY)、/dev/xxxx 或者干脆空着，空着时
# build 程序（在 tools/目录中）就使用默认值/dev/hd6。
#
# 这里/dev/hd6 对应第 2 个硬盘的第 1 个分区。这是 Linus 开发 Linux 内核时自己的机器上根
# 文件系统所在的分区位置。/dev/hd2 表示把第 1 个硬盘的第 2 个分区用作交换分区。
23 ROOT_DEV=/dev/hd6
24 SWAP_DEV=/dev/hd2
25
# 下面是 kernel 目录、mm 目录和 fs 目录所产生的目标代码文件。为了方便引用在这里将它们用
# ARCHIVES（归档文件）标识符表示。
26 ARCHIVES=kernel/kernel.o mm/mm.o fs/fs.o

# 块和字符设备库文件。'.a'表示该文件是个归档文件，也即包含有许多可执行二进制代码子程序
# 集合的库文件，通常是用 GNU 的 ar 程序生成。ar 是 GNU 的二进制文件处理程序，用于创建、修改
# 以及从归档文件中抽取文件。
27 DRIVERS =kernel/blk_drv/blk_drv.a kernel/chr_drv/chr_drv.a
28 MATH    =kernel/math/math.a          # 数学运算库文件。
29 LIBS    =lib/lib.a                   # 由 lib/目录中的文件所编译生成的通用库文件。
30
# 下面是 make 老式的隐式后缀规则。该行指示 make 利用下面的命令将所有的'.c'文件编译生成'.s'
# 汇编程序。':'表示下面是该规则的命令。整句表示让 gcc 采用前面 CFLAGS 所指定的选项以及仅使
# 用 include/目录中的头文件，在适当地编译后不进行汇编就停止（-S），从而产生与输入的各个 C
# 文件对应的汇编语言形式的代码文件。默认情况下所产生的汇编程序文件是原 C 文件名去掉'.c'后
# 再加上'.s'后缀。'-o'表示其后是输出文件的形式。其中'$*.s'（或'$@'）是自动目标变量，'$<'
# 代表第一个先决条件，这里即是符合条件'*.c'的文件。
# 下面这 3 个不同规则分别用于不同的操作要求。若目标是.s 文件，而源文件是.c 文件则会使
# 用第一个规则；若目录是.o，而原文件是.s，则使用第 2 个规则；若目标是.o 文件而原文件
# 是 c 文件，则可直接使用第 3 个规则。
31 .c.s:
32         $(CC) $(CFLAGS) \
33         -nostdinc -Iinclude -S -o $*.s $<

# 表示将所有.s 汇编程序文件编译成.o 目标文件。 整句表示使用 gas 编译器将汇编程序编译成.o
# 目标文件。-c 表示只编译或汇编，但不进行连接操作。
34 .s.o:
35         $(AS) -c -o $*.o $<
# 类似上面，*.c 文件-→*.o 目标文件。整句表示使用 gcc 将 C 语言文件编译成目标文件但不连接。
36 .c.o:
37         $(CC) $(CFLAGS) \
38         -nostdinc -Iinclude -c -o $*.o $<
39


# 下面'all'表示创建 Makefile 所知的最顶层的目标。这里即是 Image 文件。这里生成的 Image 文件
# 即是引导启动盘映像文件 bootimage。若将其写入软盘就可以使用该软盘引导 Linux 系统了。在
# Linux 下将 Image 写入软盘的命令参见 46 行。DOS 系统下可以使用软件 rawrite.exe。
40 all:    Image
41
# 说明目标（Image 文件）是由冒号后面的 4 个元素产生，分别是 boot/目录中的 bootsect 和 setup
# 文件、tools/目录中的 system 和 build 文件。42--43 行这是执行的命令。42 行表示使用 tools 目
# 录下的 build 工具程序（下面会说明如何生成）将 bootsect、setup 和 system 文件以$(ROOT_DEV)
# 为根文件系统设备组装成内核映像文件 Image。第 43 行的 sync 同步命令是迫使缓冲块数据立即写盘
# 并更新超级块。
42 Image: boot/bootsect boot/setup tools/system tools/build
```

```
43          tools/build boot/bootsect boot/setup tools/system $(ROOT_DEV) \
44                  $(SWAP_DEV) > Image
45          sync
46
    # 表示 disk 这个目标要由 Image 产生。dd 为 UNIX 标准命令：复制一个文件，根据选项进行转换和格
    # 式化。bs=表示一次读/写的字节数。if=表示输入的文件，of=表示输出到的文件。这里/dev/PS0 是
    # 指第一个软盘驱动器(设备文件)。在 Linux 系统下使用/dev/fd0。
47  disk: Image
48          dd bs=8192 if=Image of=/dev/PS0
49
50  tools/build: tools/build.c            # 由 tools 目录下的 build.c 程序生成执行程序 build。
51          $(CC) $(CFLAGS) \
52          -o tools/build tools/build.c  # 编译生成执行程序 build 的命令。
53
54  boot/head.o: boot/head.s              # 利用上面给出的.s.o 规则生成 head.o 目标文件。
55
    # 表示 tools 目录中的 system 文件要由冒号右边所列的元素生成。56--61 行是生成 system 的命令。
    # 最后的 > System.map 表示 gld 需要将连接映像重定向存放在 System.map 文件中。
    # 关于 System.map 文件的用途参见注释后的说明。
56  tools/system:   boot/head.o init/main.o \
57                  $(ARCHIVES) $(DRIVERS) $(MATH) $(LIBS)
58          $(LD) $(LDFLAGS) boot/head.o init/main.o \
59          $(ARCHIVES) \
60          $(DRIVERS) \
61          $(MATH) \
62          $(LIBS) \
63          -o tools/system > System.map
64
    # 数学协处理函数文件 math.a 由 64 行上的命令实现：进入 kernel/math/目录；运行 make 工具程序。
65  kernel/math/math.a:
66          (cd kernel/math; make)
67
68  kernel/blk_drv/blk_drv.a:           # 生成块设备库文件 blk_drv.a，其中含有可重定位目标文件。
69          (cd kernel/blk_drv; make)
70
71  kernel/chr_drv/chr_drv.a:           # 生成字符设备函数文件 chr_drv.a。
72          (cd kernel/chr_drv; make)
73
74  kernel/kernel.o:                    # 内核目标模块 kernel.o
75          (cd kernel; make)
76
77  mm/mm.o:                            # 内存管理模块 mm.o
78          (cd mm; make)
79
80  fs/fs.o:                            # 文件系统目标模块 fs.o
81          (cd fs; make)
82
83  lib/lib.a:                          # 库函数 lib.a
84          (cd lib; make)
85
86  boot/setup: boot/setup.s                        # 这里开始的三行是使用 8086 汇编和连接器
87          $(AS86) -o boot/setup.o boot/setup.s    # 对 setup.s 文件进行编译生成 setup 文件。
88          $(LD86) -s -o boot/setup boot/setup.o   # -s 选项表示要去除目标文件中的符号信息。
```

```
89
90 boot/setup.s:   boot/setup.S include/linux/config.h       # 执行 C 语言预处理，替换*.S 文
91         $(CPP) -traditional boot/setup.S -o boot/setup.s  # 件中的宏生成对应的*.s 文件。
92
93 boot/bootsect.s:        boot/bootsect.S include/linux/config.h
94         $(CPP) -traditional boot/bootsect.S -o boot/bootsect.s
95
96 boot/bootsect:  boot/bootsect.s                 # 同上。生成 bootsect.o 磁盘引导块。
97         $(AS86) -o boot/bootsect.o boot/bootsect.s
98         $(LD86) -s -o boot/bootsect boot/bootsect.o
99
   # 当执行'make clean'时，就会执行 98--103 行上的命令，去除所有编译连接生成的文件。
   # 'rm'是文件删除命令，选项-f 含义是忽略不存在的文件，并且不显示删除信息。
100 clean:
101         rm -f Image System.map tmp_make core boot/bootsect boot/setup \
102                 boot/bootsect.s boot/setup.s
103         rm -f init/*.o tools/system tools/build boot/*.o
104         (cd mm;make clean)       # 进入 mm/目录；执行该目录 Makefile 文件中的 clean 规则。
105         (cd fs;make clean)
106         (cd kernel;make clean)
107         (cd lib;make clean)
108
   # 该规则将首先执行上面的 clean 规则，然后对 linux/目录进行压缩，生成'backup.Z'压缩文件。
   # 'cd .. '表示退到 linux/的上一级（父）目录；'tar cf - linux'表示对 linux/目录执行 tar 归档
   # 程序。'-cf'表示需要创建新的归档文件 '| compress -'表示将 tar 程序的执行通过管道操作('|')
   # 传递给压缩程序 compress，并将压缩程序的输出存成 backup.Z 文件。
109 backup: clean
110         (cd .. ; tar cf - linux | compress - > backup.Z)
111         sync                                    # 迫使缓冲块数据立即写盘并更新磁盘超级块。
112
113 dep:
   # 该目标或规则用于产生各文件之间的依赖关系。创建这些依赖关系是为了让 make 命令用它们来确定
   # 是否需要重建一个目标对象。比如当某个头文件被改动过后，make 就能通过生成的依赖关系，重新
   # 编译与该头文件有关的所有*.c 文件。具体方法如下：
   # 使用字符串编辑程序 sed 对 Makefile 文件（这里即是本文件）进行处理，输出为删除了 Makefile
   # 文件中'### Dependencies'行后面的所有行，即删除了下面从 122 开始到文件末的所有行，并生成
   # 一个临时文件 tmp_make（也即 114 行的作用）。然后对指定目录下（init/）的每一个 C 文件（其实
   # 只有一个文件 main.c）执行 gcc 预处理操作。标志'-M'告诉预处理程序 cpp 输出描述每个目标文件
   # 相关性的规则，并且这些规则符合 make 语法。对于每一个源文件，预处理程序会输出一个规则，其
   # 结果形式就是相应源程序文件的目标文件名加上其依赖关系，即该源文件中包含的所有头文件列表。
   # 然后把预处理结果都添加到临时文件 tmp_make 中，最后将该临时文件复制成新的 Makefile 文件。
   # 115 行上的'$$i'实际上是'$($i) '。这里'$i'是这句前面的 shell 变量'i'的值。
114         sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
115         (for i in init/*.c;do echo -n "init/";$(CPP) -M $$i;done) >> tmp_make
116         cp tmp_make Makefile
117         (cd fs; make dep)                   # 对 fs/目录下的 Makefile 文件也作同样的处理。
118         (cd kernel; make dep)
119         (cd mm; make dep)
120
121 ### Dependencies:
122 init/main.o : init/main.c include/unistd.h include/sys/stat.h \
123   include/sys/types.h include/sys/time.h include/time.h include/sys/times.h \
124   include/sys/utsname.h include/sys/param.h include/sys/resource.h \
```

```
125    include/utime.h include/linux/tty.h include/termios.h include/linux/sched.h \
126    include/linux/head.h include/linux/fs.h include/linux/mm.h \
127    include/linux/kernel.h include/signal.h include/asm/system.h \
128    include/asm/io.h include/stddef.h include/stdarg.h include/fcntl.h \
129    include/string.h
```