专题1-工欲善其事必先利其器

/* 绿色代表引用； 紫色代表重要引用内容； 淡蓝色代表实例代码； 红色代表着重标注，可用于注意事项和重点代码。 */

一、裸机开发体验

1.1、为什么使用linux？
windows下的ADS/RVDS完成了许多重要的开发步骤。

1.2、裸机开发流程：
编写裸机程序-->调试裸机程序-->生成2进制映像-->烧写并运行2进制映像。

1.2.1、安装交叉工具链 ARM-tools

1.2.1.1 使用tar命令解压 tar -xvzf ARM-tools.tar.gz
[root@www ~]# tar [-j|-z] [cv] [-f 建立乩檔名] filename... <==打包不压缩
[root@www ~]# tar [-j|-z] [tv] [-f 建立乩檔名] <==察看檔名
[root@www ~]# tar [-j|-z] [xv] [-f 建立乩檔名] [-C 目录] <==解压缩
选项不参数：
-c ：建立打包档案，可搭配 -v 查察看过程中被打包乩档名(filename)
-t ：察看打包档案乩内容吨有哪些档名，重点在察看『档名』就是了；
-x ：解打包戒解压缩乩功能，可以搭配 -C (大写) 在特定目录解开
特别留意乩是， -c, -t, -x 丌可同时出现在一串挃令列中。
-j ：透过 bzip2 乩支持迍行压缩/解压缩：此时档名最好为 *.tar.bz2
-z ：透过 gzip 乩支持迍行压缩/解压缩：此时档名最好为 *.tar.gz
-v ：在压缩/解压缩乩过程中，将正在处理乩文件名显示出李！
-f filename：-f 后面要立刻接要被处理乩档名！建议 -f 单独写一个选项啰！
-C 目录 ：这个选项用在解压缩，若要在特定目录解压缩，可以使用这个选
项。
其他后续练习会使用到乩选项介绍：
-p ：保留备份数据乩原本权限不属性，常用于备份(-c)重要乩配置文件
-P ：保留绝对路径，亦即允讲备份数据中吨有根目录存在ㄗ意；
--exclude=FILE：在压缩乩过程中，丌要将 FILE 打包！

1.2.1.2、使用tar命令解压gcc工具链到根目录下 tar -xvzf arm-linux-gcc-4.3.2.tgz -C /

1.2.1.3、修改环境变量将工具添加到环境中 vim /root/.bashrc
export PATH=$PATH:/usr/local/arm/4.3.2/bin
保存.bashrc后需要生效 source /root/.bashrc

1.2.2、生成2进制映像
1.2.2.1、由.S文件生成.o文件 arm-linux-gcc -g -c led.S
生成led.o文件
1.2.2.2、由.o文件链接生成.elf文件(可执行文件) arm-linux-ld -Tled.lds -o led.elf led.o
生成led.o文件
1.2.2.3、由.elf文件转换生成.bin二进制文件 arm-linux-objcopy -O binary led.elf led.bin
生成led.bin文件
1.2.2.4、使用makefile生成.bin二进制文件 make
生成led.bin文件

1.2.3、烧写二进制程序到开发板
1.2.3.1、制作程序烧写SD卡
1.2.3.2、格式化NAND Flash
1.2.3.3、使用dnw下载二进制文件到开发板 (针对OK6410) /home/dnw/ led.bin 0x50008000 (0x50008000是内存，辅助安装程序会自动安装到NAND flash中）
1.2.3.4、切换NAND Flash启动

二、交叉工具链

工具汇总：交叉编译器（arm-linux-gcc） 交叉链接器（arm-linux-ld）交叉转换器（arm-linux-objcopy）交叉elf文件工具（arm-linux-readelf）交叉反汇编器（arm-linux-objdump）

一个C/C++文件需要经过预处理(preprocessing)（生成.i文件）、编译(compilation)（生成.s文件）、汇编(assembly)（生成ELF目标文件.o（OBJ文件））、连接(linking)（生成可以在特定平台运行的可执行文件）等四步才能变成可执行文件。

2.1、交叉编译器（arm-linux-gcc）

2.1.1、一步指令 arm-linux-gcc -o name name.c

分步指令：

以文件example.c为例说明它的用法

0. arm-linux-gcc -o example example.c

不加-c、-S、-E参数，编译器将执行预处理、编译、汇编、连接操作直接生成可执行代码。

-o参数用于指定输出的文件，输出文件名为example,如果不指定输出文件，则默认输出a.out

1. arm-linux-gcc -c -o example.o example.c

-c参数将对源程序example.c进行预处理、编译、汇编操作，生成example.o文件

去掉指定输出选项"-o example.o"自动输出为example.o,所以说在这里-o加不加都可以

2.arm-linux-gcc -S -o example.s example.c

-S参数将对源程序example.c进行预处理、编译，生成example.s文件

-o选项同上

3.arm-linux-gcc -E -o example.i example.c

-E参数将对源程序example.c进行预处理，生成example.i文件（不同版本不一样，有的将预处理后的内容打印到屏幕上）

就是将#include，#define等进行文件插入及宏扩展等操作。

4.arm-linux-gcc -v -o example example.c

加上-v参数，显示编译时的详细信息，编译器的版本，编译过程等。

5.arm-linux-gcc -g -o example example.c

-g选项，加入GDB能够使用的调试信息,使用GDB调试时比较方便。

6.arm-linux-gcc -Wall -o example example.c

-Wall选项打开了所有需要注意的警告信息，像在声明之前就使用的函数，声明后却没有使用的变量等。

7.arm-linux-gcc -Ox -o example example.c

-Ox使用优化选项，X的值为空、0、1、2、3

0为不优化，优化的目的是减少代码空间和提高执行效率等，但相应的编译过程时间将较长并占用较大的内存空间。

8.arm-linux-gcc -I /home/include -o example example.c

-Idirname: 将dirname所指出的目录加入到程序头文件目录列表中。如果在预设系统及当前目录中没有找到需要的文件，就到指定的dirname目录中去寻找。

9.arm-linux-gcc -L /home/lib -o example example.c

-Ldirname：将dirname所指出的目录加入到库文件的目录列表中。在默认状态下，连接程序ld在系统的预设路径中(如/usr/lib)寻找所需要的库文件，这个选项告诉连接程序，首先到-L指定的目录中去寻找，然后再到系统预设路径中寻找。

10.arm-linux-gcc –static -o libexample.a example.c

静态链接库文件

2.2、交叉链接器（arm-linux-ld）

arm-linux-ld 用于将多个目标文件，库文件连接成可执行文件。主要是“-T”选项，可以直接使用它来指定代码段，数据段，bss段的起始地址，也可以用来指定一个连接脚本，在连接脚本中进行更复杂的地址设置。“-T”选项只用于连接Bootloader，内核等“没有底层软件支持”的软件；连接运行于操作系统之上的应用程序时，无需指定“-T”选项，它们使用默认的方式进行连接。

例如： arm-linux-ld -Tled.lds -o led.elf led.o 1.o 2.o *.o

2.3、交叉elf文件工具（arm-linux-readelf）

```
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
```

```
  -n --notes            Display the core notes (if present)
  -r --relocs           Display the relocations (if present)
  -u --unwind           Display the unwind info (if present)
  -d --dynamic          Display the dynamic section (if present)
  -V --version-info     Display the version sections (if present)
  -A --arch-specific    Display architecture specific information (if any).
  -c --archive-index    Display the symbol/file index in an archive
  -D --use-dynamic      Use the dynamic section info when displaying symbols
  -x --hex-dump=<number|name>
                        Dump the contents of section <number|name> as bytes
  -p --string-dump=<number|name>
                        Dump the contents of section <number|name> as strings
  -w[liaprmfFsoR] or
  --debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=str,=loc,=Ranges]
                        Display the contents of DWARF2 debug sections
  -I --histogram        Display histogram of bucket list lengths
  -W --wide             Allow output width to exceed 80 characters
  @<file>               Read options from <file>
  -H --help             Display this information
  -v --version          Display the version number of readelf
Report bugs to <https://support.codesourcery.com/GNUToolchain/>
```

2.4、交叉反汇编器（arm-linux-objdump）

```
arm-linux-objdump -D -S led.elf > led_dump
```

2.5、交叉转换器（arm-linux-objcopy）

ARM处理器只能运行二进制文件，所以需要将可执行文件转换成二进制文件.

```
arm-linux-objcopy -O binary led.elf led.bin
```

三、Makefile 工程管理

3.1、Makefile规则

```
目标（target）... : 依赖（prerequiries）...
<tab>命令（command）
```

注意：每一行命令必须以Tab开头，不可以用空格代替。

3.2、Makefile 文件里的赋值方法

GNU make 里对变量的赋值有两种方式：延时变量，立即变量。

使用“=”、“？=”定义或者使用define指令定义的变量是延时变量。

使用“:=”定义的变量是立即变量。

3.3、Makefile 常用函数

略（后续补齐）

3.4、自动变量和通配符

3.4.1、自动变量：

“$@”表示规则的目标文件名；

“$^”表示所有依赖的名字，名字之间用空格隔开；

“$<”表示第一个依赖的文件名。

3.4.2、通配符

“%”是通配符，它和一个字符串中任意个数的字符相匹配。

3.4.3、“$”的使用

“$”本属于函数应用，这里这讲述$(origin variable)，用于查询变量的名称。

3.5、伪目标

只包含命令，却没有依赖的目标，例如clean

```
.PHONY: clean
clean:
    rm -f hello main.o func.o
```

“.PHONY”将“clean”目标声明为伪目标。

3.6、最终目标

3.6.1、当一个makefile中有多条规则时，如何单独执行某条 规则？

make 目标

例如 make led.o

3.6.2、如果用户没有指定执行某一条规则，make会默认 执行makefile中的第1条规则，而这条规则中的目标称之为：最终目标。

3.7、去回显

命令前面加“@”可以去掉回显。

3.8、修改makefile名

用 make -f filename 指明使用的makefile工程名。

四、链接器脚本

4.1、链接器脚本的作用

链接器脚本用于反映代码段，数据段，bss段的信息。

4.2、脚本构成

```
1 OUTPUT_ARCH(arm)
2 ENTRY(_start)
3 SECTIONS {
4       . = 0x50008000;
5
6       . = ALIGN(4);
7       .text
8       {
9               start.o(.text)
10              *(.text)
11      }
12
13      . = ALIGN(4);
14      .data
15      {
16              *(.data)
17      }
18
19      . =ALIGN(4);
20      bss_start = . ;
21      .bss
22      {
23              *(.bss)
24      }
25      bss_end = . ;
26 }
```

4.2.1、链接脚本的基本命令是SECTIONS命令，它描述了输出文件的“映射图”：输出文件中各段，个文件怎么放置。一个SECTIONS命令内部包含一个或多个段，段（Secion）是链接脚本的基本单元，它表示输出文件中的某部分怎么放置。

4.2.2、设置输出设备类型和入口

第1行 OUTPUT_ARCH(arm) 是用来设置输出设备类型，第2行 ENTRY(_start) 用于设置程序入口。

4.2.3、设置起始链接地址

第4行 . = 0x50008000; 设置起始链接地址。

4.2.4、对齐设置

虽然指定了运行地址，但考虑到大多数ARM设备是以4字节操作，所以可以使用ALIGN(align)来指定对齐的要求，这个对齐的地址才是真正的运行地址，使用方式如 . = ALIGN(4);

4.2.5、变量

在链接脚本中可以通过变量来保存一下参数，如第20行 bss_start = . ; 和第25行 bss_end = . ; 虽然都是将当前地址保存到变量中，但因为中间隔着bss段，所以这两个变量的值是不一样的。

4.2.6、代码段首文件

```
7       .text
8       {
9               start.o(.text)
10              *(.text)
11      }
```

这段代码定义了一个名为“.text”的段，它的内容为“*(.text)”,表示所有输入文件的代码段。这些代码被集合在一起，其实运行地址为0x50008000，在第9行插入 start.o(.text) 是指明代码段首文件为start.o。

五、Eclipse集成开发环境

5.1、安装集成开发环境

5.1.1、格式化开发板NAND Flash

5.1.2、安装gdb server

解压安装，并添加环境变量，且要保证环境变量在arm-linux-工具之前。

5.1.3、安装jlink软件

测试失败

5.1.4、安装Eclipse

5.2、使用Eclipse建立裸机代码工程并调试