

《嵌入式系统》实验指导书

冯迅 编著

云南师范大学信息学院

2016 年 7 月

目 录

实验一	Linux 基本命令	1
实验二	嵌入式系统开发环境的搭建.....	7
实验三	裸机跑单灯（LED）闪烁实验（可选）	25
实验四	Linux 下程序的编译及调试	26
实验五	程序的交叉编译及移植.....	29
实验六	Bootloader 定制实验（可选）	31
实验七	嵌入式 Linux 内核的配置、裁减及编译	32
实验八	嵌入式根文件系统的制作.....	35
实验九	嵌入式 Linux 下的进程控制实验	38
实验十	嵌入式 Linux 下的文件编程实验	51
实验十一	嵌入式 Linux 内核模块开发实验	60
实验十二	嵌入式 Linux 下的点灯（LED）实验.....	64
实验十三	嵌入式 Linux 下的 GUI 实验（可选）	70

实验一 Linux 基本命令

一、实验目的

- 1、熟悉 Linux 系统环境，学会使用命令行进行操作。
- 2、掌握 Linux 的基本命令。
- 3、掌握 Linux 下的文本编辑器 vi 的使用方法。

二、实验设备

PC 机、CentOS6.6 系统。

三、实验内容

- 1、启动 Linux 并以 root 用户身份登录，进入终端命令行模式，注销、重启、关闭 Linux。

步骤：（注：以下输入的字符中不包含双引号）

- （1）登录名称输入“root”并回车。
- （2）输入 root 用户的登录密码并回车，注意输入的密码不会显示在屏幕上，所以输入时一定要确保正确，否则不能登录。
- （3）登录后的命令行提示符为“#”号，表示登录用户为 root 用户，输入“logout”并回车后即可登出（也可直接按快捷键 Ctrl+D）；
- （4）登录后在命令行中输入“reboot”并回车，可重新启动 Linux 系统；
- （5）登录后在命令行中输入“poweroff”并回车，可关闭 Linux 系统。

- 2、在终端命令行中进行以下命令的操作：ls、clear、cd、pwd、mkdir、touch、rm、cp、mv、chmod、cat、less、tar。

步骤：（注：以下输入的字符中不包含双引号）

- （1）列表命令（ls）：
 - A、在命令行中输入“ls”并回车，可查看当前目录下的所有文件及目录；
 - B、在命令行中输入“ls -l”并回车（也可输入“ll”并回车），可查看当前目录下所有文件的详细信息；
 - C、在命令行中输入“ls install.log”并回车，可单独查看当前目录下名为 install.log 的文件，一般用来确认当前目录是否存在该文件。
 - D、在命令行中输入“ls -l install.log”并回车，可单独查看当前目录下名为 install.log 文件的详细信息。

- E、在命令行中输入“`ls /`”并回车，可指定查看根目录下的所有文件及目录；
 - F、在命令行中输入“`ls /root/.bash_profile`”并回车，可显示 root 目录下名为.bash_profile 的文件，虽然文件为隐藏形式也可显示出来，而直接使用“`ls /root`”命令是看不到隐藏文件的。
 - G、在命令行中输入“`ls /root/in*`”并回车，可显示 root 目录下以字母 in 开头的所有文件。
- (2) 清屏命令 (clear)：输入“`clear`”并回车，可进行清屏操作。
- (3) 更改目录命令 (cd)：
- A、在命令行中输入“`cd /`”并回车，可返回到根目录；
 - B、在命令行中输入“`cd /home`”可切换到 home 目录下；
 - C、在命令行中输入“`cd ..`”可返回上一级目录；
 - D、在命令行中输入“`cd /var/ftp/pub`”可一次性切换到 pub 目录下；
 - E、在命令行中输入“`cd /root`”可切换到 root 用户目录下（注：root 用户目录显示为“`~`”）；
 - F、目录操作可使用绝对路径（从根目录开始），也可使用相对路径（从当前目录开始）；为了能快速准确地更改目录，目录操作应善于使用 TAB 键进行目录名称的自动补齐操作。
- (4) 查看当前路径命令 (pwd)：输入“`pwd`”并回车，可显示当前目录所在的绝对路径。
- (5) 创建目录命令 (mkdir)：
- A、在命令行中输入“`mkdir abc`”并回车，可在当前目录下创建一个名为 abc 的新目录；
 - B、在命令行中输入“`mkdir a b c`”并回车，可在当前目录下同时创建多个目录；
 - C、在命令行中输入“`mkdir /123`”并回车，可在指定目录（根目录）下创建新目录 123；
 - D、在命令行中输入“`mkdir -p /a/b/c`”并回车，可在根目录下创建出三个嵌套的目录 a/b/c；
 - E、注意，新创建的目录名称必须在创建目录下不存在，即不能存在两个名称一样的目录。
- (6) 创建文件命令 (touch)：
- A、在命令行中输入“`touch 123`”并回车，可新建名为 123 的文件；
 - B、在命令行中输入“`touch a b c`”并回车，可同时新建多个文件；

C、注意，新创建的文件名称必须在创建目录下不存在，即不能存在两个名称一样的文件。

(7) 删除命令 (rm) :

- A、假设当前目录下存在一个名为 123 的文件，在命令行中输入“rm 123”并回车，可删除该文件，但需要按 y 键确认；
- B、假设当前目录下存在一个名为 123 的文件，在命令行中输入“rm -f 123”并回车，可删除该文件，此时为强行删除，不需要确认；
- C、假设当前目录下存在一个名为 abc 的目录，在命令行中输入“rm -r abc”并回车，可删除该目录，但需要按 y 键确认；
- D、假设当前目录下存在一个名为 abc 的目录，在命令行中输入“rm -fr abc”并回车，可删除该目录，此时为强行删除，不需要确认；
- E、在命令行中输入“rm -f a*”并回车，可强行删除当前目录下以字母 a 开头的所有文件，不需要确认；
- F、注意，要删除文件或目录前首先要保证它们都存在，不可能删除一个不存在的对象。

(8) 拷贝命令 (cp) :

- A、假设当前目录下存在一个名为 123 的文件，在命令行中输入“cp 123 /var/ftp/pub”并回车，可把名为 123 的文件拷贝到 pub 目录下；
- B、在拷贝时还可进行更名，在命令行中输入“cp 123 /var/ftp/pub/456”并回车，则拷贝文件 123 到 pub 目录下的同时被更名成 456；
- C、若拷贝的是目录，则要加上参数“-r”，在命令行中输入“cp -r /var/ftp/pub /home”并回车，则把 pub 目录拷贝到了 home 下；
- D、注意，要拷贝文件或目录时首先要保证它们存在，不能拷贝一个不存在的对象。

(9) 移动命令 (mv) :

- A、假设当前目录下存在一个名为 123 的文件，在命令行中输入“mv 123 /var/ftp/pub”并回车，可把名为 123 的文件移动到 pub 目录下；
- B、移动时还可进行更名，在命令行中输入“mv 123 /var/ftp/pub/456”并回车，则文件 123 移动到 pub 目录下的同时把它更名为了 456；

- C、移动目录时可进行同样的操作，不需要加参数，在命令行中输入“`mv /var/ftp/pub /home`”并回车，则把 `pub` 目录移动到了 `home` 下；
- D、注意，要移动文件或目录前首先要保证它们存在，不可能移动一个不存在的对象；
- E、另外，还可使用 `mv` 命令来对文件或目录进行更名，在命令行中输入“`mv 123 456`”并回车，则当前目录下的文件 `123` 被更名为 `456`。

(10) 更改权限命令 (`chmod`)：

- A、假设当前目录下存在一个名为 `abc` 的文件，先使用命令“`ls -l abc`”来查看它的权限；
- B、在命令行中输入“`chmod 777 abc`”并回车，对文件 `abc` 的权限进行更改；
- C、再使用命令“`ls -l abc`”来查看是否更改成功；
- D、其中 4 代表“只读”，2 代表“可写”，1 代表“可执行”，数字可相加进行权限组合，如 7 代表“可读可写可执行”，而 777 代表所有用户都对该文件都拥有“可读可写可执行”的权限。

(11) 查看文本内容命令 (`cat/less`)：

- A、在命令行中输入“`cat /etc/passwd`”并回车，可查看文本文件 `passwd` 的内容，但只能查看最后一页，只适合查看一屏以内的小文本文件；
- B、在命令行中输入“`less /etc/passwd`”并回车，可使用上下光标键及上下翻页键来滚动查看文本文件 `passwd` 的全部内容，特别适合用于查看内容较多的大文本文件，查看完毕后按 `q` 键退出。

(12) 解压命令 (`tar`)：

- A、假设当前目录下存在一个名为 `abc.tar.gz` 的压缩文件，可在命令行中输入“`tar -zxvf abc.tar.gz`”并回车来进行解压；
- B、若压缩文件的后缀为 `bz2`，则要输入“`tar -jxvf abc.tar.bz2`”来进行解压。

3、在终端命令行中启动 `vi`，并进行文本的添、删、查、改以及保存、退出等基本操作。

步骤：（注：以下输入的字符中不包含双引号）

(1) 创建文本文件：

- A、假设当前目录下不存在名为 `abc` 的文件，则输入“`vi abc`”并回车，可在当前目录下新建一个名为 `abc` 的文本文件，进入 `vi` 编

- 辑器后，按一次“I”键进入插入模式（屏幕底端显示 INSERT），此时可录入内容，在此录入一句“Hello world!”；
- B、录入完成后按一次“Esc”键返回到命令状态，然后输入一个冒号，此时屏幕底行会出现一个“:”，在其后面输入“wq”并回车，可对文件进行存盘退出，若不想保存，则输入“q!”并回车直接强行退出。
- (2) 打开文本文件：接着第(1)步创建的文本文件，在命令行中输入“vi abc”并回车，可打开该文件，在屏幕上可看到刚才输入的内容。
- (3) 删除文本内容：
- A、为了方便实验，先拷贝一个系统中已有的文本文件来进行，执行命令“cp /etc/passwd /root/test”；
- B、执行命令“vi /root/test”，使用 vi 打开它；
- C、文件打开后，其内容显示在屏幕上，要对其中某一行进行删除，可在命令状态下（即非插入模式下），把光标定位到想要删除的行（可使用上下光标键），然后连续按两次“D”键即可删除该行；
- D、要删除多行，可在按 D 键前加入要删除的行数，比如要连续删除 7 行，可把光标定位到要删除内容的首行，然后连续按“7DD”即可；
- E、要删除从光标位置开始至本行结尾的内容，可在定位好光标后，连续按“D\$”即可。
- (4) 拷贝文本内容：
- A、执行命令“vi /root/test”，使用 vi 打开文本文件 test；
- B、要拷贝某一行的内容，在命令状态下，把光标定位到想要拷贝的行，然后连续快速按两次“Y”键，随后把光标定位到想要粘贴的地方，然后按一次“P”键即可完成拷贝；
- C、要进行多行拷贝，可在按 YY 前加入要拷贝的行数，比如要连续拷贝 6 行，可把光标定位到要拷贝内容的首行，然后再连续按“6YY”，随后把光标定位到想要粘贴的地方，然后按一次“P”键即可完成多行拷贝，粘贴可进行多次；
- (5) 编辑文本内容：
- A、执行命令“vi /root/test”，使用 vi 打开文本文件 test；
- B、文本打开后，按一次“I”键进入插入模式（屏幕底端显示 INSERT），此时可使用键盘上的光标键来定位光标位置，利用上下翻页键来进行文本查看；

- C、要增加文本内容，可把光标定位在要加入的位置，然后通过键盘录入内容即可；
 - D、要删除某些内容，可利用光标键、删除键及退格键来完成；
 - E、编辑操作完成后，参照第（1）步进行存盘退出的操作。
- （6）字符串查找：
- A、执行命令“**vi /root/test**”，使用 vi 打开文本文件 test；
 - B、在底行模式下输入“**/var**”并回车，则可查找到文本中自光标处开始向下遭到的第一个 var 字符串，找到后光标会移动到该行；
 - C、要继续查找，按“**N**”键即可；
 - D、要向上查找，把“**/**”符号换成“**?**”即可，其实在键盘上“**?**”号在上，所以是向上查找，“**/**”在下，所以是向下查找。
- （7）字符串替换：
- A、执行命令“**vi /root/test**”，使用 vi 打开文本文件 test；
 - B、在底行模式下输入“**s/var/abc**”并回车，可把本行中的第一个 var 替换为 abc；
 - C、在底行模式下输入“**s/var/abc/g**”并回车，可把本行中的所有 var 替换为 abc；
 - D、在底行模式下输入“**1,20s/var/abc/g**”并回车，可把 1~20 行中的所有 var 替换为 abc。
- （8）设置行号：
- A、执行命令“**vi /root/test**”，使用 vi 打开文本文件 test；
 - B、在底行模式下（在命令状态下输入一个冒号），输入“**set nu**”并回车，可显示出行号，便于查找操作，行号本身并不会加入到实际的文本文件中；
 - C、要取消行号显示，可在底行模式下输入“**set nonu**”并回车即可。
- （9）以上是使用 vi 进行的基本操作，其实在 Linux 系统中还可以使用 vim，它的功能与 vi 一致，但在文本的显示上使用了颜色来加以区分，特别适合程序编辑。

实验二 嵌入式系统开发环境的搭建

一、实验目的

- 1、掌握嵌入式系统开发环境的搭建方法。
- 2、掌握相关工具软件的使用方法。

二、实验设备

- 1、PC 机。
- 2、相关工具软件。
- 3、嵌入式系统实验箱。
- 4、相关通信电缆。
- 5、仿真器（选）。

三、实验内容

- 1、在 PC 机上安装虚拟机，推荐使用 VMWare。

步骤：

- (1) 对 VMWare 进行常规安装，这里选用 9.0.2 的版本，若没有特殊要求对安装选项均可采用默认值进行；
- (2) 启动 VMWare，其程序界面如图 3.1-1 所示；

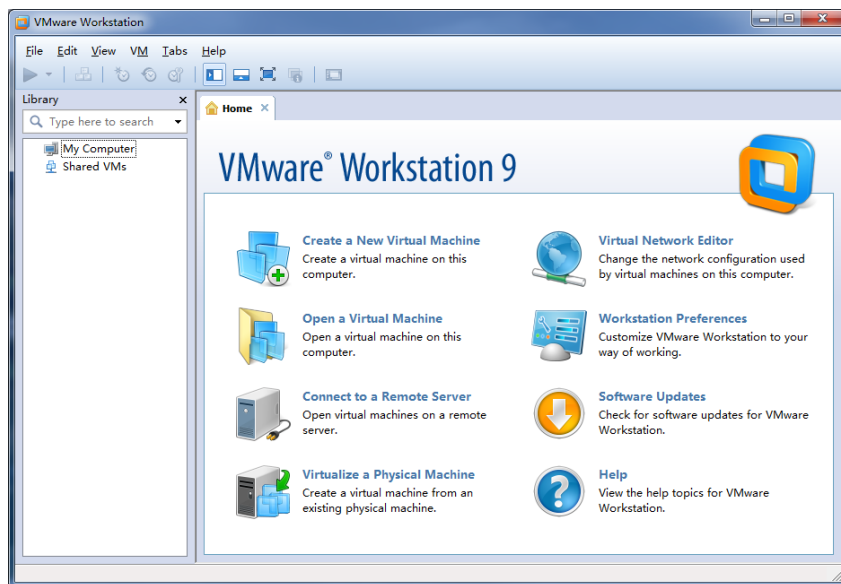


图 3.1-1

- (3) 点击其中的第一项“Create a New Virtual Machine”，创建一个新的虚拟机系统，在弹出的对话框中选择“Typical(recommended)”推荐选项，点击“Next”按钮；

- (4) 接下选择安装来源，可从光盘或系统映像进行安装，这里先不进行安装，选择“I will install the operating system later”，点击“Next”按钮；
- (5) 在弹出的对话框中要求选择要安装的操作系统，这里选择“Linux”，并在 Version 中选择“CentOS”，点击“Next”按钮；
- (6) 接下来问你虚拟机的名称及文件存储位置，这里名称定为“CentOS”，路径可取默认值，设置好后点击“Next”按钮；
- (7) 在弹出的对话框中设置硬盘容量，这里可视 PC 机硬盘容量而定，一般可取 20G，然后在下面选择“Split virtual disk into multiple files”一项，把镜像文件分成多个文件，以提高文件系统的兼容性，点击“Next”按钮；
- (8) 在弹出的对话框中点击“Customize Hardware”按钮，进行机器硬件的自定义配置，对不需要的硬件可把它 Remove 掉，只留下有用的，具体可参考图 3.1-2 进行；

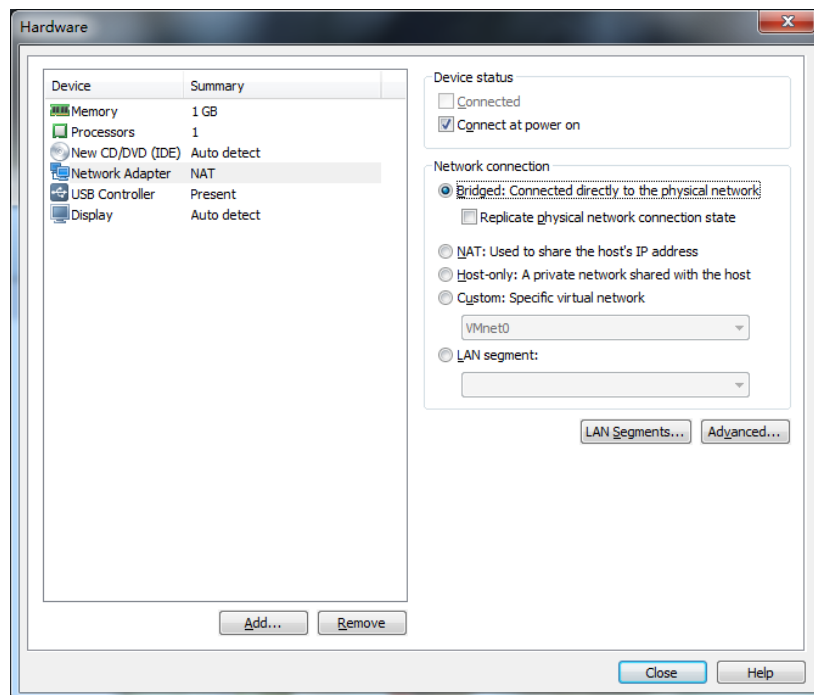


图 3.1-2

- (9) 特别要注意的是网络配置，必须选择“Bridged”选项，否则后面可能会出现网络不通的问题，设置完成后点击“Close”按钮关闭对话框，最后点击“Finish”按钮关闭配置对话框；
- (10) 接下来点击菜单 Edit->Virtual Network Editor 打开虚拟网络编辑的对话框，其内容只保留图 3.1-3 所示的项目，其余的全部移除掉；

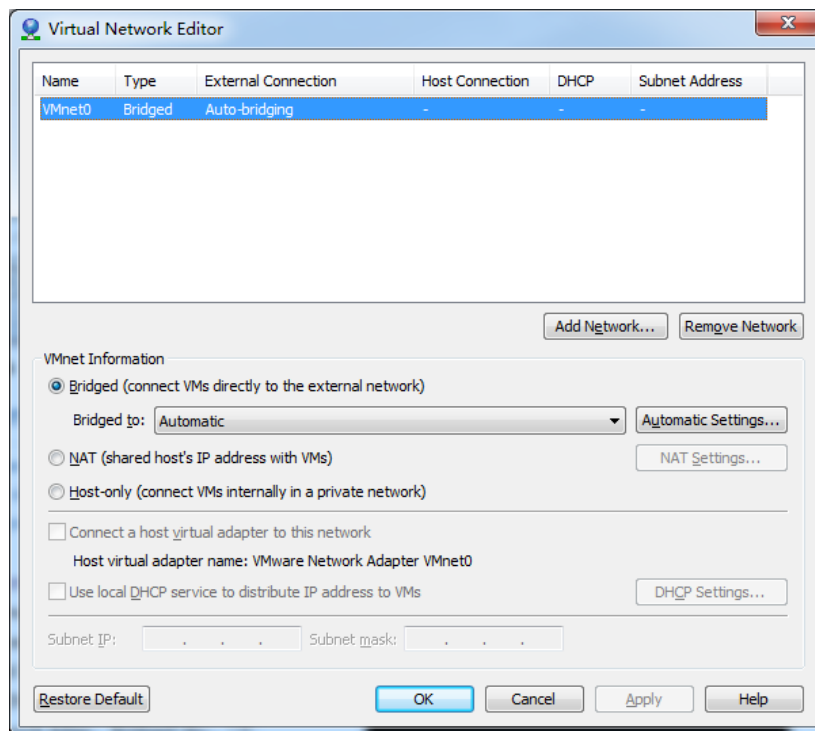


图 3.1-3

(11) 点击“OK”按钮结束整个虚拟机的配置。

2、在 VM 中安装 Linux，并对其进行配置以保证能顺利使用，推荐使用 CentOS。

步骤：

- (1) 这里使用 CentOS6.6 的 Linux 版本进行安装,为了节约电脑系统资源,保证 Linux 顺畅运行,这里只安装字符界面 (Level 3), 不安装桌面环境;
- (2) 安装可通过光盘或映像文件进行,放入 CentOS6.6 的第 1 张光盘 (或载入映像文件), 启动刚才配置好的 CentOS 虚拟机 (点击 “Power on this virtual machine”), 过一会儿进入安装 Linux 的欢迎界面, 用上下光标键选中第一项 “Install or upgrade an existing system”, 然后回车继续 (注: 按 Ctrl+G 进入虚拟机界面, 按 Ctrl+Alt 返回 PC 机);
- (3) 随后进入安装盘检测界面, 这里不用检测, 按 “Skip” 键跳过 (注: 用 Tab 键选择, 回车键确认);
- (4) 在过一会儿出现的界面中, 用鼠标点击 “Next” 继续;
- (5) 接下来选择安装时的提示语言, 这个无所谓, 这里就用默认的英文, 点击 “Next” 继续;
- (6) 接下来选择键盘, 这里就用默认的美式键盘, 点击 “Next” 继续;
- (7) 接下来选择安装的存储装置, 默认选择 “Basic Storage Devices”, 点击 “Next” 继续;
- (8) 接下来会出现警告, 点击 “Yes,discard any data” 继续;

- (9) 接下来问你主机名称，一般不用管它，用默认值即可，点击“Next”继续；
- (10) 接下来选择时区，这里选择“Asia/Shanghai”，点击“Next”继续；
- (11) 接下来要你输入 root 密码，密码不能为空，这里就输入“123456”即可，点击“Next”后会出现警告，说密码太简单，不用管它，点击“Use Anyway”继续；
- (12) 接下来是硬盘分区，这里选择自定义分区，即“Create Custom Layout”一项，点击“Next”继续；
- (13) 在接下来的界面中，先点击“Create”按钮新建分区，在弹出的对话框中默认选择“Standard Partition”一项，点击“Create”弹出新建分区对话框，这里先新建 boot 分区，配置可参照图 3.2-1 进行，完成后点击“OK”按钮退出；

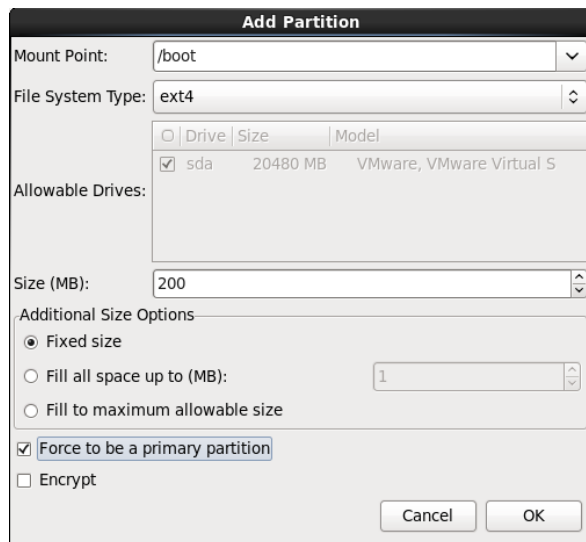


图 3.2-1

- (14) 接下来再点击“Create”按钮新建交换分区，同样选择“Standard Partition”一项，点击“Create”弹出新建分区对话框，交换分区的配置可参照图 3.2-2 进行，完成后点击“OK”按钮退出；
- (15) 接下来再点击“Create”按钮新建根分区，同样选择“Standard Partition”一项，点击“Create”弹出新建分区对话框，根分区的配置可参照图 3.2-3 进行，完成后点击“OK”按钮退出；
- (16) 全部分区配置完成后的样子如图 3.2-4 所示，确认后点击“Next”按钮继续；
- (17) 接下来是格式化警告，点击“Format”按钮对分区进行格式化；
- (18) 再次出现警告，点击“Write changes to disk”继续；
- (19) 格式化完成后会出现 Boot loader 选择对话框，此处全部按默认值，点击“Next”按钮继续；

Add Partition

Mount Point: <Not Applicable>

File System Type: swap

Allowable Drives:

Drive	Size	Model
<input checked="" type="checkbox"/> sda	20480 MB	VMware, VMware Virtual S

Size (MB): 512

Additional Size Options

☒ Fixed size

☐ Fill all space up to (MB): 512

☐ Fill to maximum allowable size

☒ Force to be a primary partition

☐ Encrypt

Cancel OK

图 3.2-2

Add Partition

Mount Point: /

File System Type: ext4

Allowable Drives:

Drive	Size	Model
<input checked="" type="checkbox"/> sda	20480 MB	VMware, VMware Virtual S

Size (MB): 200

Additional Size Options

☐ Fixed size

☐ Fill all space up to (MB): 1

☒ Fill to maximum allowable size

☒ Force to be a primary partition

☐ Encrypt

Cancel OK

图 3.2-3

CentOS - VMware Workstation

File Edit View VM Help

CentOS

Please Select A Device

Device	Size (MB)	Mount Point/ RAID/Volume	Type	Format
Hard Drives				
sda (vdev100)				
sda1	200	/boot	ext4	✓
sda2	512		swap	✓
sda3	19767	/	ext4	✓

Create Edit Delete Reset

Back Next

To direct input to this VM, click inside or press Ctrl+G.

图 3.2-4

- (20) 接下来出现安装模块选项对话框，这里全部采用自定义方式，点击选择“Customize now”一项，然后点击“Next”按钮继续；
- (21) 接下来弹出自定义选择对话框，如图 3.2-5 所示，左边为分类，右边为具体的安装模块；

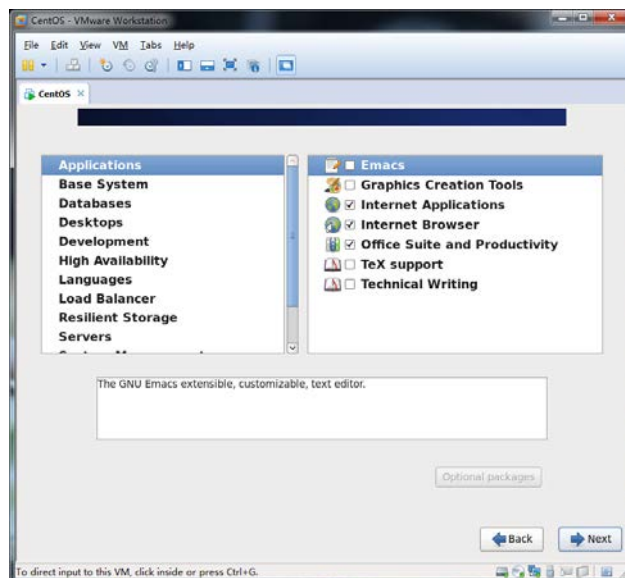


图 3.2-5

- (22) 在“Base System”分类中，勾选 Base、Compatibility libraries、Console internet tools、Debugging Tools、Directory Client、Network file system client、Networking Tools 等项，在“Development”分类中，勾选 Additional Development、Development tools、Server Platform Development 等项，在“Server”分类中，勾选 CIFS file server、Directory Server、FTP server、NFS file server、Network Infrastructure Server、Server Platform、System administration tools 等项，其余分类及项目均不用勾选，点击“Next”按钮继续；
- (23) 接下来系统会进行软件安装，需要耐心等待一定时间，结束后点击“Reboot”按钮重启系统；
- (24) 重启完成后以 root 身份登录进入，输入“ifconfig”并回车，可看到当前还未给网卡配置 ip 地址，如图 3.2-5 所示；
- (25) 在命令行下输入“cd /etc/sysconfig/network-scripts”并回车进入网络配置目录，输入“vi ifcfg-eth0”并回车打开网卡配置文件，把其中的内容改成如图 3.2-6 所示的形式；
- (26) 修改时要注意大小写，其中的第二行 HWADDR 等号后面是网卡的 MAC 地址，要按实际值填写不能照抄，其实在刚打开的文件中就有这一句，不用更改它即可；

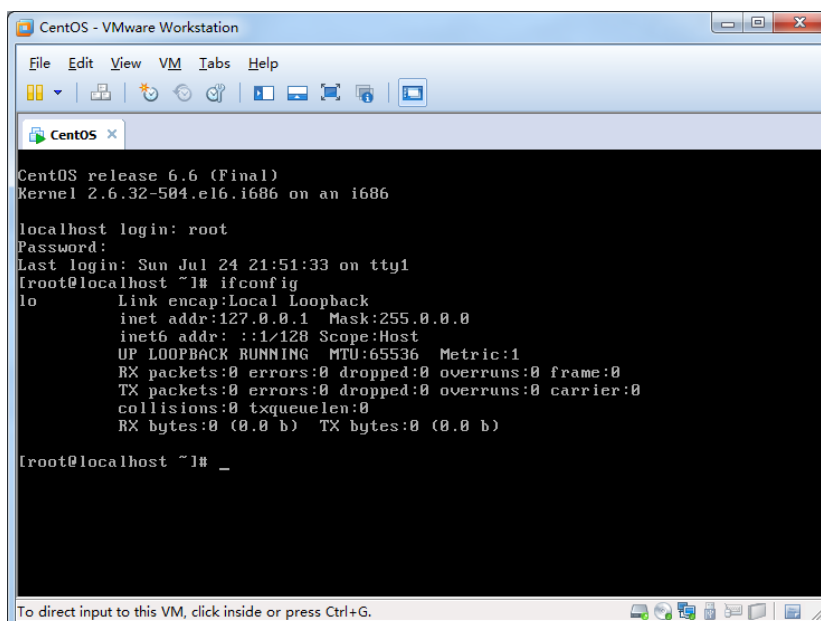


图 3.2-5

```

DEVICE=eth0
HWADDR=00:0C:29:21:61:3E
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=static
IPADDR=192.168.0.200
NETMASK=255.255.255.0

```

图 3.2-6

- (27) 完成后存盘退出，然后输入 **reboot** 重启一次系统；
 - (28) 重启完成后再次执行上面的第 24 步，可以看到多出了一项 **eth0**，表明已经给网卡 **eth0** 绑定了一个静态的 **ip** 地址 **192.168.0.200**；
 - (29) 接下来要关闭防火墙，在命令行下输入“**ntsysv**”并回车，用上下光标键找到 **ip6tables** 和 **iptables** 两项，把前面的星号去掉(利用空格键)，然后点击“**OK**”（利用 **Tab** 键）退出；
 - (30) 接下来要禁用 **SELINUX**，在命令行下输入“**vi /etc/selinux/config**”并回车打开配置文件，在其中找到 **SELINUX=enforcing** 这一项，把它改成 **SELINUX=disabled**，然后存盘退出；
 - (31) 最后输入 **reboot** 再重启一次系统，Linux 就配置完成了。
- 3、在 PC 上安装相关工具软件，如 **tftpd**、**8uftp**、**SecureCRT**、**SourceInsight**、**ARM-MDK** 等，并进行配置以保证能顺利使用。
- 步骤：（注：进行网络操作时最好关掉 PC 机端的防火墙）

- (1) 常规安装 Tftpd 软件（也可使用绿色免安装版），注意如果操作系统是 64 位的，尽量安装 Tftpd64；
- (2) Tftpd 安装完成后运行它，点击“Browse”按钮选定下载目录，同时在“Server interfaces”选项中选择有线网络（即连接实验箱的网络）接口的 ip 地址，其它选项采用默认设置，如图 3.3-1 所示；

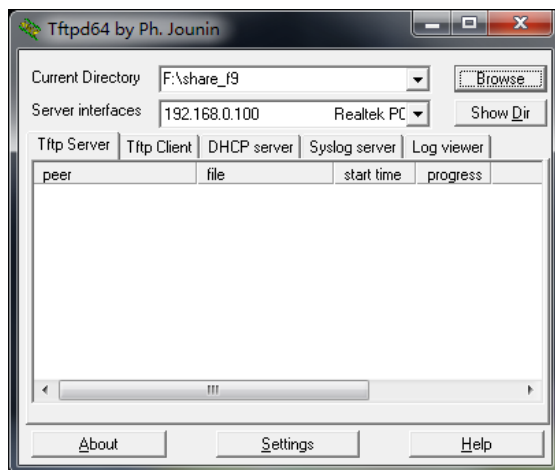


图 3.3-1

- (3) Tftpd 软件提供了临时的 tftp 服务器功能，在整个服务期间不要关闭它，在传输文件时，软件会显示出相关信息。
- (4) 8uftp 为一款绿色免安装软件，提供 ftp 的客户端服务功能，可用于连接虚拟机或实验箱，运行软件后界面如图 3.3-2 所示；
- (5) 在地址栏中输入 ftp 服务端的 ip 地址，用户名是要访问的 ftp 用户帐户，密码要输入正确，端口默认为 21，点击“连接”按钮，连接成功后在“远程”选项卡窗体中会显示出服务器端的 ftp 目录情况；

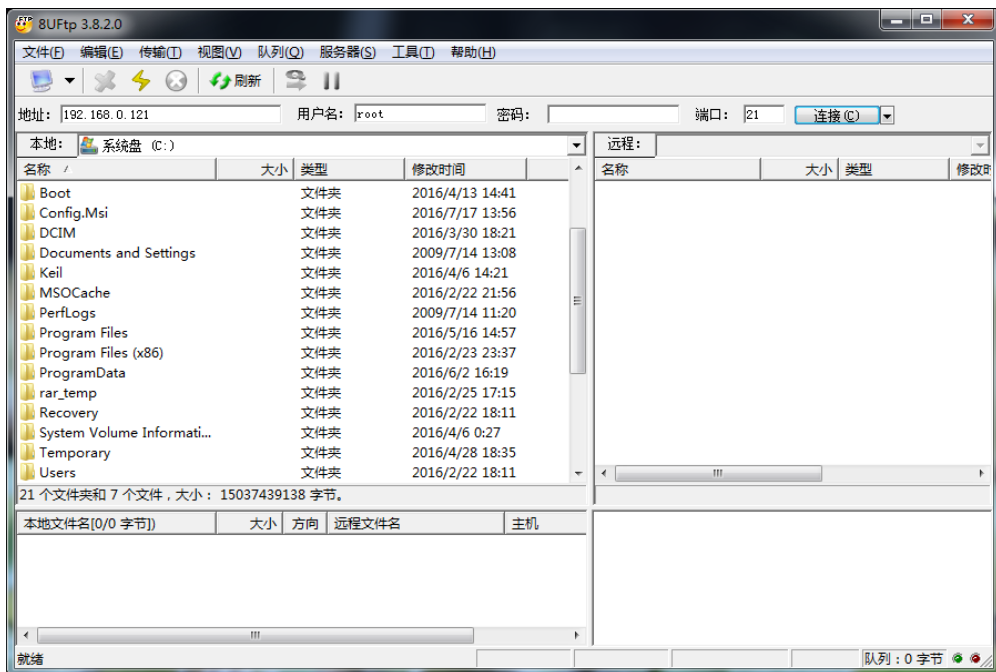


图 3.3-2

- (6) 连接后可直接在“本地”选项卡窗体和“远程”选项卡窗体之间拖曳文件或目录。
- (7) SecureCRT 软件提供了多种终端连接功能，在嵌入式系统开发中常用它来提供串口的终端服务，同时也可用它来实现 SSH 的远程连接服务；
- (8) 常规安装 SecureCRT 并运行它，图 3.3-3 是它首次运行时的界面；

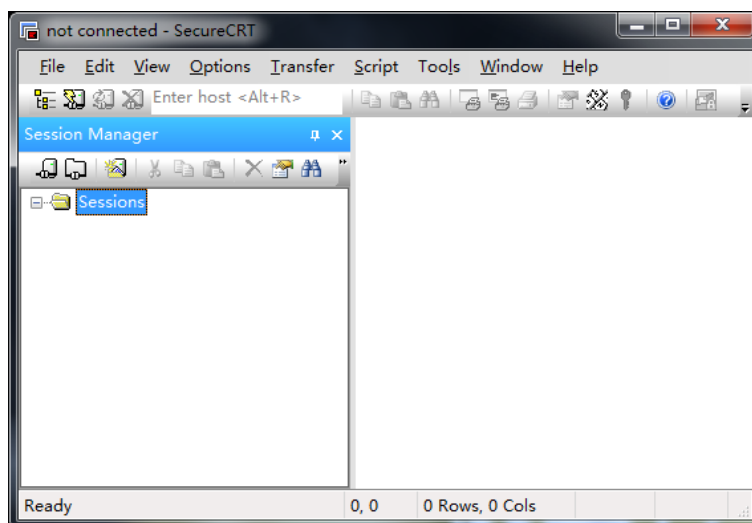


图 3.3-3

- (9) 在工具条上点击第二个按钮“Quick Connect”（或直接按 Alt+Q），弹出 Quick Connect 对话框，此处按图 3.3-4 中所示进行设置；

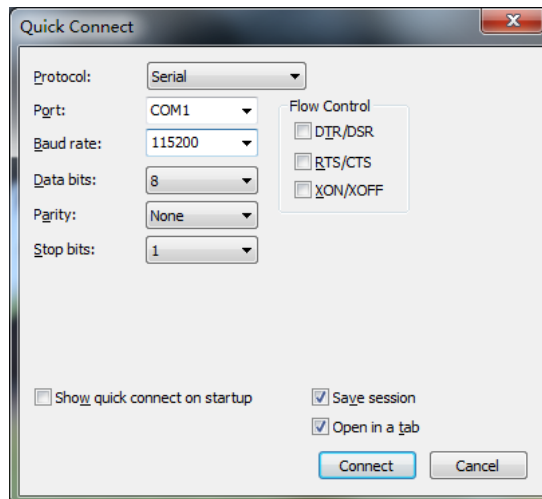


图 3.3-4

- (10) 点击“Connect”按钮后，就连接到了实验箱上，显示实验箱中运行的 Linux，并可远程进行操作，如图 3.3-5 所示；
- (11) 以后运行 SecureCRT，只需要双击左边的“serial-com1”标签就可连接上实验箱，然后回一下车就可以看到内容了；
- (12) SecureCRT 的 SSH 连接请自行研究配置。

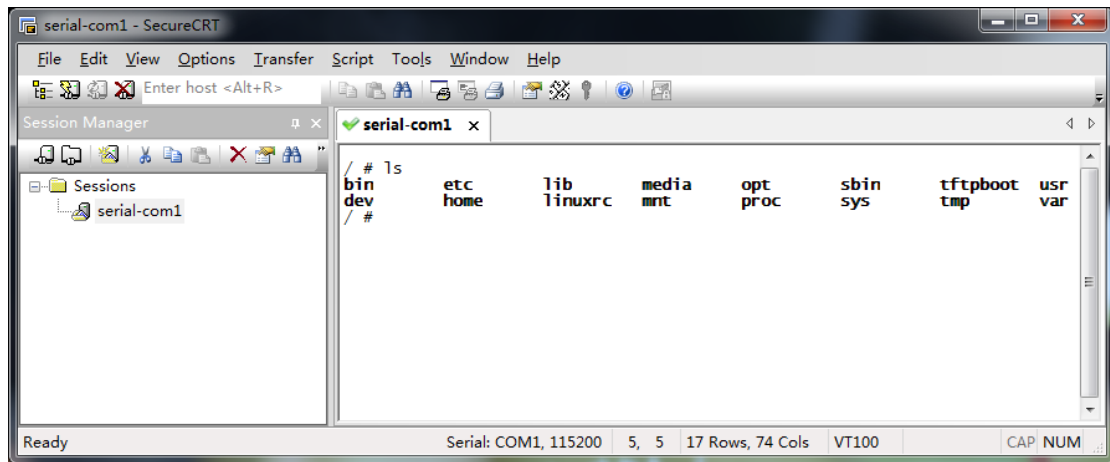


图 3.3-5

- (13) SourceInsight 是一款功能强大的编辑软件,这里主要用它来管理 Linux 的内核源代码,方便查询和编辑;
- (14) 常规安装 SourceInsight 并运行它,图 3.3-6 是它首次运行时的界面;

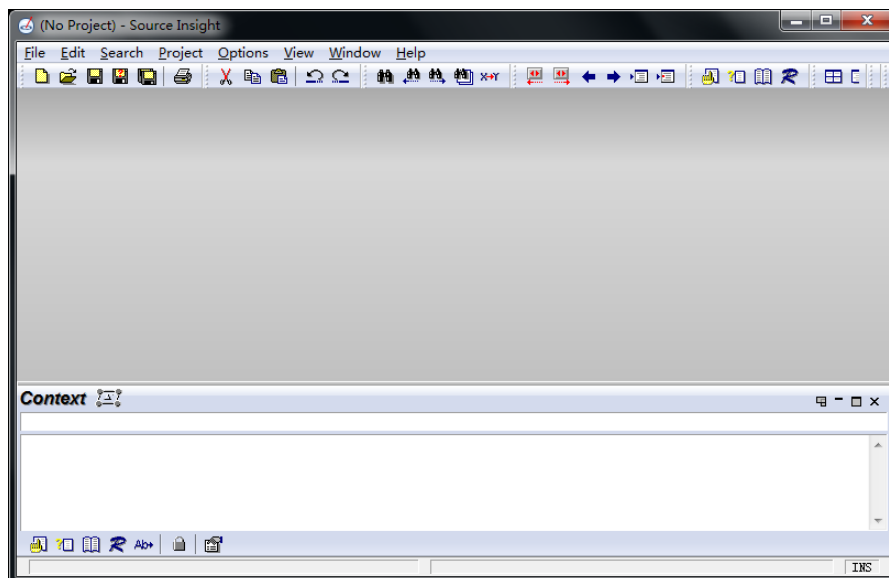


图 3.3-6

- (15) 点击菜单 **Project->New Project** 创建一个工程,在弹出的对话框中输入工程名称(如 MyLinux),点击“OK”按钮;
- (16) 在弹出的“New Project Settings”对话框中,在“Configuration”部分可以选择工程使用全局配置文件还是自己单独的配置文件,这里最好选择单独的配置文件,其余部分采用默认值即可,点击“OK”按钮;
- (17) 随后会弹出“Add and Remove Project Files”对话框,让你选择需要的文件添加到 Project 中,通过左边的树状图中找到要添加的文件,点击“Add”按钮就可以了(如图 3.3-7),可以把不通路径下的文件都添加到同一个工程中,而不用拷贝源文件,完后点击“Close”按钮关闭对话框;

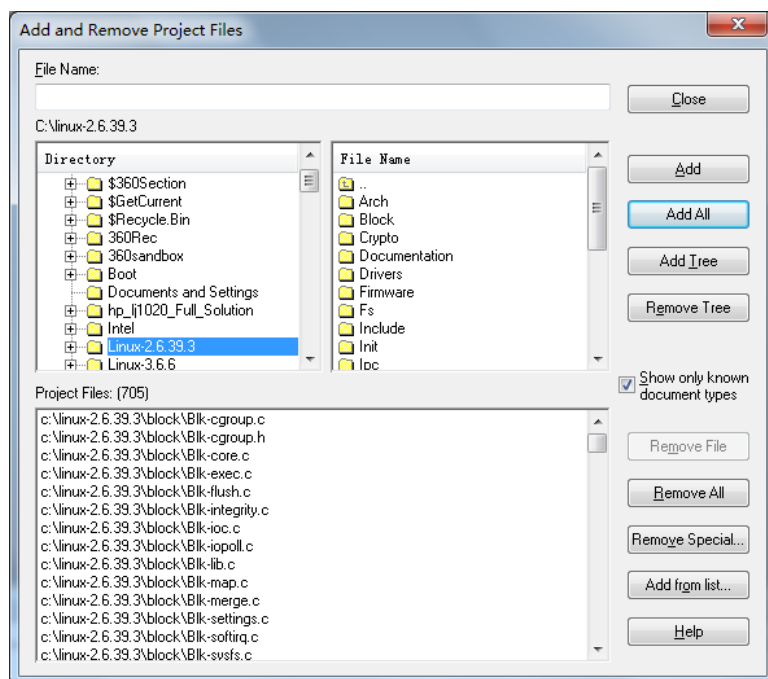


图 3.3-7

(18) 导入文件后，在右边的列表中双击你要查看的程序文件，即可展开文件进行显示，大致如图 3.3-8 所示；

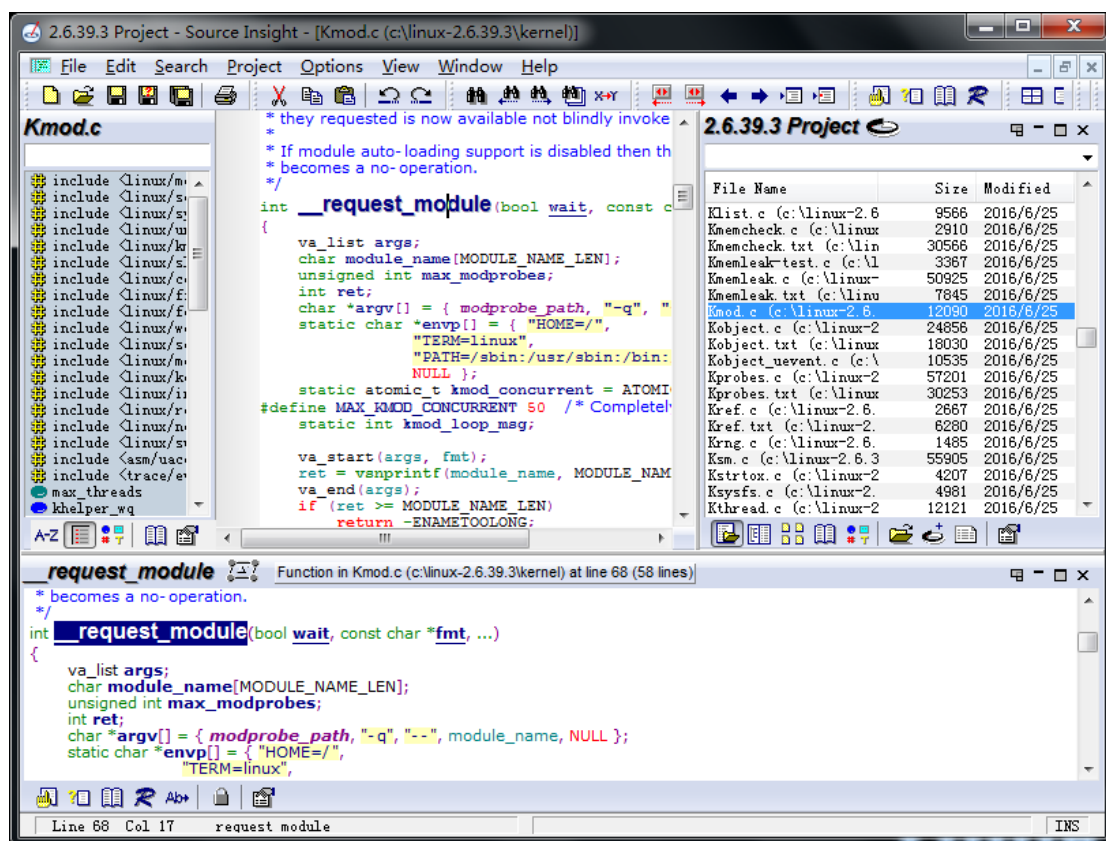


图 3.3-8

- (19) 为了建立文件之间的关联，需要同步一下整个工程的文件，点击菜单 Project->Synchronize Files，在弹出的对话框中勾选需要的选项，然后点击“OK”按钮，首次进行会需要一定的时间；
- (20) SourceInsight 功能非常强大，要用好它请花时间自行研究。
- (21) ARM-MDK 软件主要用来开发裸机程序，这里选择 uVision4 的版本，常规安装 ARM-MDK 并运行它，图 3.3-9 是它首次运行时的界面；

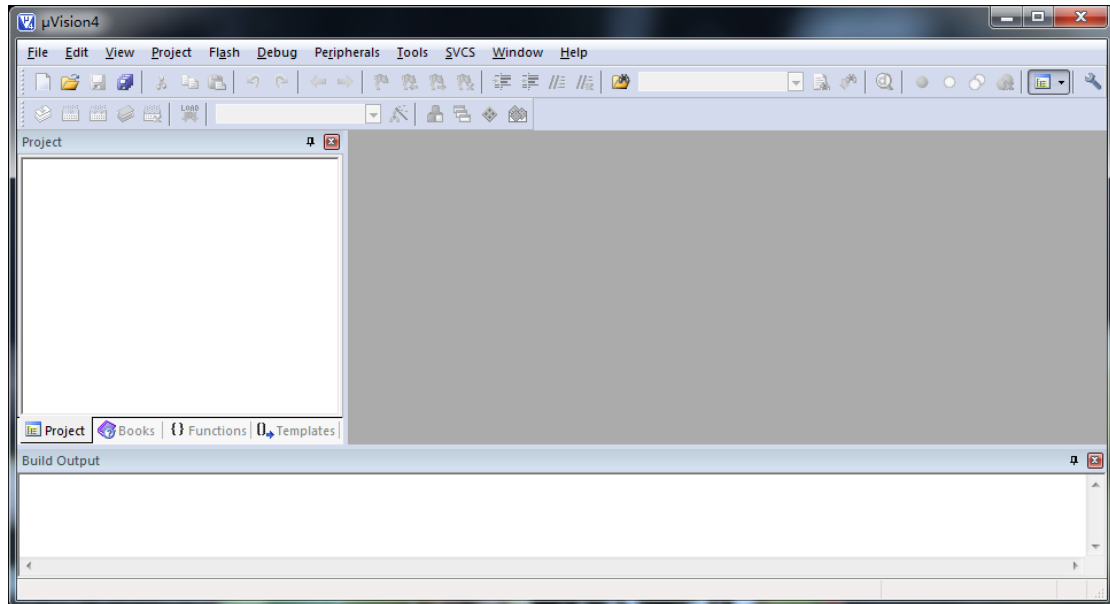


图 3.3-9

- (22) 点击菜单 Project->New uVision Project 创建一个工程，在弹出的对话框中给工程选择路径并输入一个名称（例如 first），点击“保存”按钮；
- (23) 在弹出的对话框中要求选择开发器件，找到“Samsung”并展开它，选择 S3c2410A，点击“OK”按钮，如图 3.3-10 所示；

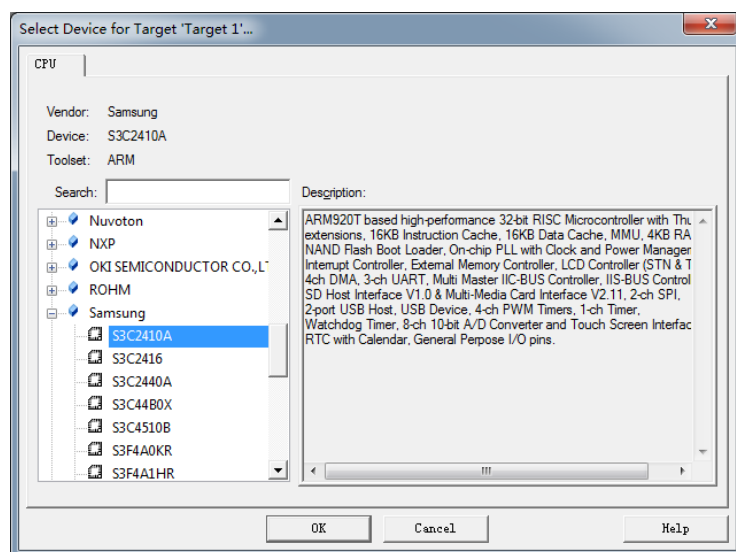


图 3.3-10

- (24) 随后会询问是否要拷贝启动代码到工程中，点击“是”同意；
 - (25) 点击菜单 **File->New**（或点击工具条上的“New”按钮），新建一个程序文本文件，在其中输入程序代码，然后点击菜单 **File->Save**（或点击工具条上的“Save”按钮），在弹出的对话框中输入程序文本的名称（注意要加上扩展名），如 `test.c`；
 - (26) 展开左边的“Target 1”，在下面的“Source Group 1”上双击弹出“Add Files to Group ‘Source Group 1’”对话框，定位到刚才程序文件保存的目录，找到 `test.c` 文件，点击“Add”按钮把文件加入到工程中去（注意只能点击一次），然后点击“Close”按钮退出；
 - (27) 在左边的“Source Group 1”下双击刚才导入进来的文件 `test.c`，这样就进入到程序的编辑状态了；
 - (28) 点击工具条上的“Options for Target”按钮（或直接按 `Alt+F7`）打开一个配置对话框，切换到 **Output** 标签页勾选上“Create HEX file”及“Create Batch file”选项，以生成十六进制及二进制的烧写文件；
 - (29) 切换到 **User** 标签页，在“Run User Programs After Build/Rebuild”一栏的 **Run #1:**后面输入“`fromelf.exe --bin -o ./first.bin ./first.axf`”（注意，此处的 `first` 应该与实际的生成文件名称一致，默认生成文件名称为工程文件名称），并勾上其前面的复选框；
 - (30) 通过上面的设置，就可以让 Keil 生成的 `axf` 格式的二进制文件转换成 `bin` 格式，以支持 S3C2410 的下载；
 - (31) 切换到 **Target** 标签页，在这里可根据实际需要，对 **ROM** 区和 **RAM** 区的地址进行设置，以适应应用程序的个体要求。
- 4、连接 PC 机和嵌入式系统实验箱的相关线路，如网线、串口通信线、并口线以及仿真器等。
- 步骤：
- (1) 利用并口电缆线连接好 PC 机与实验箱并上紧；
 - (2) 利用串口电缆线连接好 PC 机与实验箱并上紧；
 - (3) 利用对调网线连接好 PC 机与实验箱并插紧；
 - (4) 把仿真器的 USB 端利用 USB 线连接到 PC 机的 USB 口，JTAG 端利用排线连接到实验箱；
 - (5) 以上第 1 和第 4 项应根据具体情况选择是否有必要进行。
- 5、在 PC 机与嵌入式系统实验箱之间用 FTP 进行文件传输，用串口终端进行控制操作；在 PC 机与虚拟机之间用 Samba 进行共享通讯；在虚拟机与嵌入式系统实验箱之间用 NFS 进行共享通讯。
- 步骤：（注：以下输入的字符中不包含双引号）

- (1) PC 机与实验箱之间通过 FTP 进行文件传输,这里利用 8uftp 软件来实现, 详见上面项目 3 中的第 (4) ~ (6) 步。
- (2) 用串口终端进行控制操作, 这里利用 SecureCRT 软件来实现, 详见上面项目 3 中的第 (7) ~ (11) 步。
- (3) 在 PC 机与虚拟机之间用 Samba 进行共享通讯,需要在虚拟机的 Linux 中进行 Samba 服务配置,首先要确认 Linux 中安装了 Samba 服务并能开机自启动,在命令提示符下输入“ntsysv”并回车,找到“smb”选项并确认已勾选上(利用空格键),如图 3.5-1 所示;

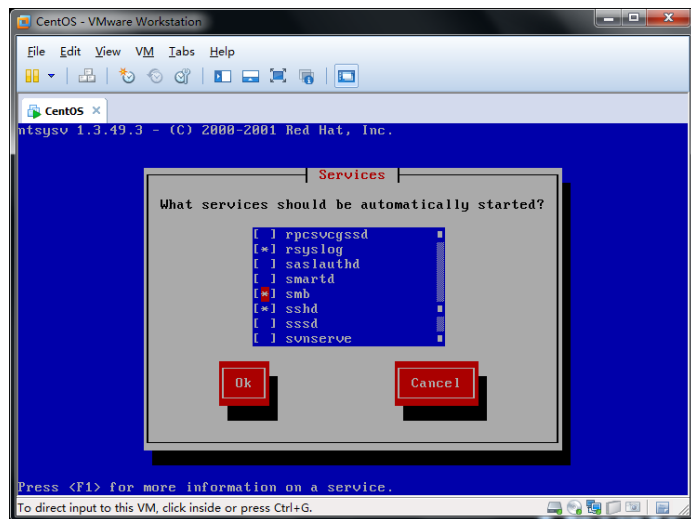


图 3.5-1

- (4) 返回到命令提示符,输入“mkdir /share”并回车,建立 Samba 共享目录,输入“chmod 777 /share”并回车,让 share 目录拥有所有权限;
- (5) 在命令提示符下输入“cd /etc/samba”并回车,进入到 Samba 服务的配置目录,用 vi 打开配置文件 smb.conf,在 “[global]” 这项下面找到“workgroup = MYGROUP”这句,把它改成“workgroup = WORKGROUP”,并确保这一行没有被注释掉,然后回车再加入一行,内容为“null passwords = yes”;
- (6) 再往下查找到“security = user”这句,把它改成“security = share”,同样要确保这一行没有被注释掉;
- (7) 继续往下找,大约快到结尾处会找到像“#==== Share Definitions =====”这样的一行,其下面有[homes]、[printers]、[netlogon]、[Profiles]、[public]等项目内容,把这些内容全部注释掉(也可全部删掉),然后在末尾处增加一段如图 3.5-2 所示的内容,最后存盘退出;


```
[share]
comment = share
path = /share
public = yes
writable = yes
browseable = yes
available = yes
guest ok = yes
```

图 3.5-2

- (8) 重启一次 Samba 服务，输入“service smb restart”并回车即可；
- (9) 在 PC 机上打开运行对话框（可直接按“徽标+R”键），输入“\\192.168.1.200\share”并回车即可打开 Linux 下的 share 目录（注：此处的 ip 地址应为 Linux 的实际 ip 地址）；
- (10) 若访问不了或提示没有权限访问，很可能是防火墙没关闭，windows 的要关闭，Linux 下的也要关闭（详见项目 2 中的（29）、（30）步）；
- (11) 成功后可在 Windows 下把它映射成一个网络盘符，以后就可以随意使用了。
- (12) 在虚拟机与实验箱之间用 NFS 进行共享通讯，需要在虚拟机的 Linux 中进行 NFS 服务配置，首先要确认 Linux 中安装了 NFS 服务并能开机自启动，在命令提示符下输入“ntsysv”并回车，找到“nfs”选项并确认已勾选上(同时还要确保勾选上 netfs、nfslock 及 network 选项)，如图 3.5-3 所示；

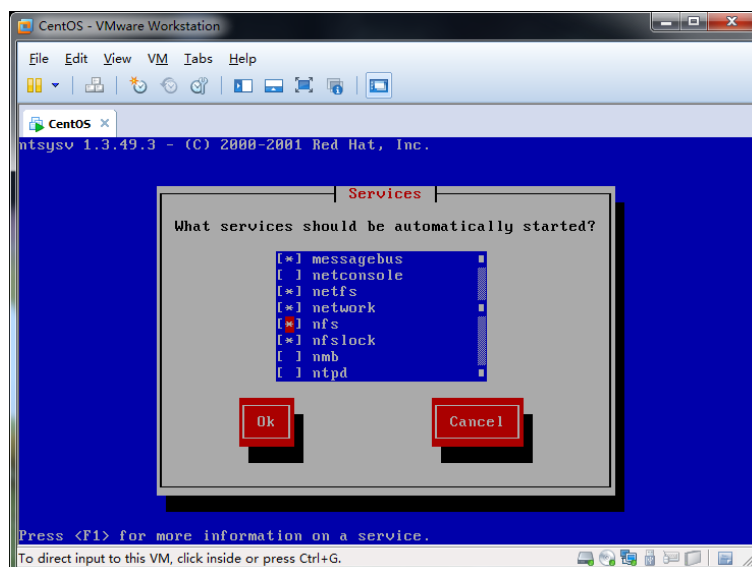


图 3.5-3

- (13) 在虚拟机的 Linux 中，输入“vi /etc/exports”并回车，打开 NFS 的配置文件 exports，该文件默认是空的，加入如图 3.5-4 所示的一行内容，把“/share”目录作为 NFS 的共享目录，存盘退出；

```
/share *(rw,sync,no_root_squash)
```

图 3.5-4

- (14) 重启一次 NFS 服务, 输入 “service nfs restart” 并回车即可;
- (15) 在实验箱的 Linux 命令提示符下, 输入如图 3.5-5 所示的命令后回车, 若成功则把实验箱下的 “/mnt/cdrom” 目录挂载到了虚拟机里的 “/share” 目录下 (注: 图中的 ip 地址应为虚拟机中 Linux 的实际 ip 地址);

```
mount -t nfs -o nolock, rsize=4096 192.168.0.200:/share /mnt/cdrom
```

图 3.5-5

- (16) 此时在 PC 机的网络盘符下 (或虚拟机里的 “/share” 目录下) 拷入一个文件, 则在实验箱的 “/mnt/cdrom” 下即可看到该文件了。

6、对嵌入式系统实验箱进行 Bootloader、Kernel 及根文件系统的烧写, 并进行相关配置以保证其能顺利使用。

步骤: (注: 本例程在 XP 系统下进行, 以下输入的字符中不包含双引号)

- (1) 烧写 Bootloader, 这里选用并口进行 vivi 的烧写, 需要用仿真器进行烧写的请自行研究;
- (2) 首先连接好并口电缆, 然后安装并口驱动程序, 先把整个 GIVEIO 目录 (在实验箱配套光盘的 Linux/img/flashvivi 目录下) 拷贝到 c:\windows 目录下, 然后把该目录下的 giveio.sys 文件拷贝到 c:\windows\system32\drivers 下;
- (3) 打开控制面板, 选添加硬件, 点击 “下一步” 按钮, 在弹出的对话框中选择 “是, 我已经连接了此硬件” 一项, 然后点击 “下一步” 按钮; 在弹出的对话框中选择最末尾的 “添加新的硬件设备” 一项, 然后点击 “下一步” 按钮; 在弹出的对话框中选择 “安装我手动从列表选择的硬件” 一项, 然后点击 “下一步” 按钮; 在弹出的对话框中选择最上面的 “显示所有设备” 一项, 然后点击 “下一步” 按钮; 在弹出的对话框中选择 “从磁盘安装” 一项, 然后点击 “下一步” 按钮; 在弹出的对话框中点击 “浏览” 按钮, 指定目录到 c:\windows\GIVEIO\giveio.inf 文件, 点击 “打开” 按钮, 然后点击 “确定” 按钮; 随后点击 “下一步” 至 “完成” 按钮即安装好并口驱动。
- (4) 在 C 盘下新建一个名为 bootloader 的目录, 然后把 sjf2410-s.exe 和 vivi 两个文件 (在实验箱配套光盘的 Linux/img/flashvivi 目录下) 拷贝进去;
- (5) 确保实验箱电源已打开, 在 windows 的命令行下进入到 c:\bootloader 目录下, 输入命令 “sjf2410-s /f:vivi” 并回车, 此后会出现三次要求输入参数的过程, 三次均选择 0 并回车, 随后就进入 vivi 的烧写过程,

此过程出以字符 p 为进度进行显示，烧写需要一定的时间，中途不允许断电或执行其它非法操作，烧写完成后选择 2 并回车退出。

- (6) 接下来烧写内核，确保 PC 机与实验箱的网线和串口线已连接好，打开实验箱电源进入到 vivi 状态下，启动 SecureCRT 软件并连接到实验箱进行操作；
- (7) 在 vivi 状态下依次执行三条命令：“ifconfig ip 192.168.0.121”、“ifconfig server 192.168.0.100”、“ifconfig save”；
- (8) 在 PC 机上启动 Tftpd 软件，并把下载目录指向到内核文件 zImage 所在的目录（实验箱配套光盘的 Linux/img 目录），并确保 PC 机的 ip 地址为 192.168.0.100；
- (9) 在实验箱 vivi 下输入命令“tftp flash kernel zImage”并回车，若一切正常会出现下载进度并很快完成，下载后会自动对内核进行烧写，需要等待一定的时间，确保烧写完成后可重启实验箱（可按 Reset 键），若烧写正常则过一会儿就可以看到 Linux 内核启动了。
- (10) 由于还未烧写根文件系统，所示启动的 Linux 进入不了命令行模式，重启实验箱并按任意键再次进入到 vivi 状态，输入命令“tftp flash root root.cramfs”并回车进行根文件系统的下载及烧写，完成后重启实验箱即可进入 Linux 命令行。
- (11) 在实验箱上进入 Linux 后，输入命令“ifconfig”并回车，会发现还没有网络，依次输入两条命令：“ifconfig eth0 192.168.0.121”、“ifconfig inetd”即可，这样就可以通过该 ip 地址来进行通讯了；
- (12) 在 PC 机上可以通过 ftp 的方式来访问实验，访问的 ip 地址为 192.168.0.121，用户为 root，密码为空。

7、Linux 命令行下使用 U 盘等外部存储器。

步骤：（注：本例中 U 盘为 FAT32 形式，以下输入的字符中不包含双引号）

- (1) 确保当前状态在虚拟机的 Linux 中，在 PC 上插入 U 盘，此时 Linux 会显示一些信息，不用管它直接回一下车就可看见命令提示符了；
- (2) 在命令行下输入“fdisk -l”并回车，查看系统为 U 盘分配的设备名称，如图 3.7-1 所示；
- (3) 查看上图中最末一行，U 盘被定的设备名称为 sdb1，接下来要挂载这个设备，在命令行下输入“mount -t vfat /dev/sdb1 /mnt/cdrom”并回车，若没有异常情况，就可以进入/mnt/cdrom 目录来访问 U 盘了；
- (4) 使用完毕后，需要反挂载（即卸载）U 盘才能拔出，先在命令行下输入“cd /”并回车返回到根目录（不允许在 U 盘的目录下进行卸载），而后执行命令“umount /dev/sdb1”后即可拔出 U 盘了；

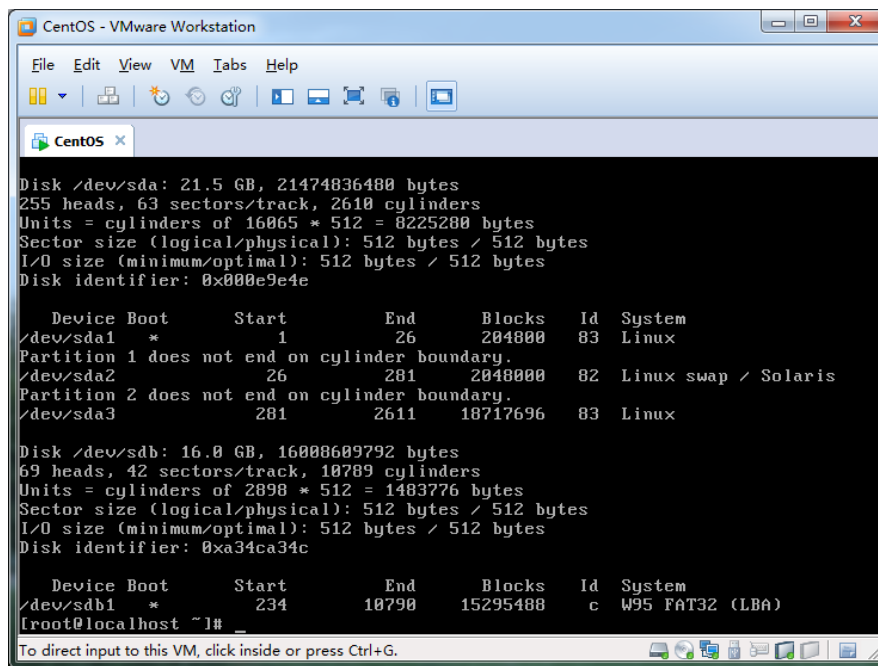


图 3.7-1

- (5) 用鼠标点击虚拟机右下角处的倒数第三个图标（指向它会显示出 U 盘名称），可让 U 盘在虚拟机和 PC 机之间进行切换。

实验三 裸机跑单灯（LED）闪烁实验（可选）

一、实验目的

- 1、掌握 ARM 处理器 I/O 口的控制方法。
- 2、掌握裸机程序的开发方法。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、根据嵌入式系统实验箱的结构,选用其中接有 LED 的一个 I/O 口,分析其“亮/灭”的条件,并确定对应的 I/O 口电平状态。

步骤:

- 2、配置 ARM 处理器相应的 I/O 口,并对其输出电平进行控制。

步骤:

- 3、延时程序可采用消耗机器指令周期方式编写,也可使用定时器方式。

实验四 Linux 下程序的编译及调试

一、实验目的

- 1、掌握在 Linux 下进行程序开发的方法。
- 2、掌握 gcc 和 gdb 的使用方法。
- 3、掌握 Makefile 的使用方法。

二、实验设备

PC 机。

三、实验内容

- 1、启动 vi 并建立一个 hello.c 的程序文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在 “/home” 目录下新建一个名为 test 的目录，进入 test 目录，把所有实验的文件都放到该目录下；（①mkdir /home/test ②cd /home/test）
- （2）在该目录下利用 vi 新建一个名为 hello.c 的文件；（vi hello.c）

- 2、在 hello.c 中编写一个能在屏幕上打印出 “Hello World!” 的 C 程序。

步骤：

- （1）先在 vi 中按 “i” 键进入插入模式；
- （2）在屏幕中输入如图 2.2-1 所示内容；

```
#include <stdio.h>
int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

图 2.2-1

- （3）完成后存盘退出。

- 3、利用 gcc 对 hello.c 进行编译并生成 hello 文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在命令提示符下输入 “gcc hello.c -o hello” 并回车，编译 hello.c 文件，若有错误则根据提示进行修改，直到编译通过；
- （2）用 “ls -l” 命令查看是否生成了 hello 文件，并查看其权限。

- 4、对 hello 文件增加可执行属性，并运行它。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 执行命令“`chmod 777 hello`”，把 `hello` 文件修改为拥有所有权限；
- (2) 输入“`./hello`”并回车，查看程序运行结果。

5、尝试用 `gdb` 对应用程序进行调试。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 利用 `vi` 新建一个名为 `gdb_test.c` 的文件，其内容如图 2.2-2 所示；

```
#include <stdio.h>
int main(void)
{
    int i;
    long result = 0;
    for(i=1;i<=100;i++)
        result += i;
    printf("result=%d\n",result);
    return 0;
}
```

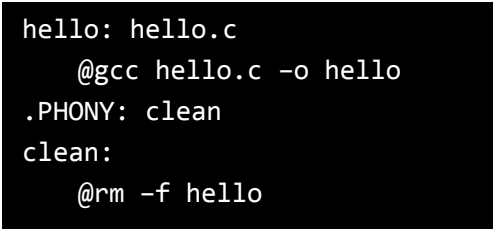
图 2.2-2

- (2) 在命令提示符下输入“`gcc -g gdb_test.c -o gdb_test`”并回车，编译生成可调试的执行文件 `gdb_test`；
- (3) 输入“`gdb gdb_test`”并回车，进入 `gdb` 调试环境；
- (4) 在 `gdb` 提示符下输入“`break main`”（或“`b main`”）并回车，在 `main` 函数入口处打上一个断点；
- (5) 输入“`run`”（或“`r`”）并回车，运行调试程序，可观察到运行停在了 `main` 函数的断点处；
- (6) 输入“`next`”（或“`n`”）并回车，继续单步运行调试程序（不进入子函数）；
- (7) 输入“`step`”（或“`s`”）并回车，继续单步运行调试程序（进入子函数）；
- (8) 输入“`print result`”（或“`p result`”）并回车，可查看当前变量 `result` 的值；
- (9) 输入“`continue`”（或“`c`”）并回车，继续运行调试程序直到结束，并查看运行结果；
- (10) 输入“`list`”（或“`l`”）并回车，可查看程序内容；
- (11) 输入“`info break`”并回车，可查看已打的断点情况；
- (12) 输入“`delete 1`”并回车，可删除已打的第 1 个断点；
- (13) 输入“`quit`”（或“`q`”）并回车，可退出 `gdb` 调试环境；
- (14) 学有余力的同学还可自行实验条件断点如“`break 7 if i=10`”，及变量监控如“`watch i`”等调试功能。

6、利用 Makefile 工程管理形式重复上述第 3 项。

步骤：（注：以下输入的字符中不包含双引号）

- （1）先重复第 1、2 项，编写好 hello.c 文件；
- （2）再利用 vi 建立一个名为 Makefile 的文件并与 hello.c 放在同一个目录下，Makefile 文件的内容如图 2.2-3 所示；



```
hello: hello.c
    @gcc hello.c -o hello
.PHONY: clean
clean:
    @rm -f hello
```

图 2.2-3

- （3）在该目录下输入命令“make”并回车，若没有错误则会生成文件 hello，若有错误则根据提示进行修改，直到生成 hello 文件；
- （4）执行命令“chmod 777 hello”，把 hello 文件修改为拥有所有权限；
- （5）输入“./hello”并回车，查看程序运行结果；
- （6）输入“make clean”并回车，可删除刚才生成的文件 hello。

实验五 程序的交叉编译及移植

一、实验目的

- 1、了解 ARM 的交叉编译工具链。
- 2、掌握程序的交叉编译过程。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、创建“实验二”中的 `hello.c` 文件。

步骤：（略，请参见“实验二”进行）

- 2、利用 `arm-linux-gcc` 对 `hello.c` 进行交叉编译并生成 `hello` 文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）交叉编译工具要根据所使用的 Linux 内核版本来选择，一般 2.4.X 的 Linux 内核选择 2.X.X 的版本，而 2.6.X 的 Linux 内核选择 3.X.X 的版本，这里由于实验箱的 Linux 是 2.4 的版本，所以选择的交叉编译工具为 2.95.2 版本；
- （2）安装交叉编译工具链，这里使用第三方制作好的，使用时直接用 `tar` 命令解压即可，此处把 2.95.2 版的交叉编译工具解压到目录 `/opt/host/armv4l` 中；
- （3）把交叉编译工具链所在的目录添加到搜索路径中，在命令提示符下输入“`vi ~/.bashrc`”并回车打开该文件，在末尾加入一句“`export PATH = /opt/host/armv4l/bin:$PATH`”（该句也可加在“`~/.bash_profile`”文件中），存盘退出；
- （4）为使刚才添加的搜索路径生效，可有三种方式选择，一是执行“`reboot`”命令重启 Linux，二是先执行“`logout`”登出再重新登录，三是执行命令“`source ~/.bashrc`”；
- （5）进入到刚才编辑的 `hello.c` 所在的目录，执行“`armv4l-unknown-linux-gcc hello.c -o hello`”并回车，即可生成基于 ARM 平台的可执行文件 `hello`。

- 3、利用工具把生成的 `hello` 文件下到嵌入式系统实验箱中。

步骤：

- (1) 把上面虚拟机中编译好的文件 `hello` 拷贝到 `/share` 目录下，并在 PC 机端的网络盘符下确认找到它；
- (2) 确保 PC 机与实验箱之间网线已连通，在 PC 机端运行程序 `8uftp`，在地址栏中输入实验箱的 ip 地址 `192.168.0.121`，用户名为 `root`，密码为空，点击“连接”按钮，稍等一会儿后会显示欢迎信息，表示已连通；
- (3) 在 `8uftp` 的“本地”选项卡窗体中定位到网络盘符，在“远程”选项卡窗体中定位到要拷贝到的目录，用鼠标把网络盘符中的 `hello` 文件直接拖曳到远程目标目录下，待拷贝完成后，文件 `hello` 就传输到了实验箱中；
- (4) 如果采用了 NFS 方式，则更为方便，只要文件 `hello` 拷贝到了虚拟机的 `/share` 目录下，就可在实验箱的 `/mnt/cdrom` 下找到它了，具体操作可参见“实验三”中第 5 项的第 (11) ~ (15) 步。

4、对 `hello` 文件增加可执行属性，并运行它。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 通过 `SecureCRT` 操作实验箱中的 `Linux`，进入上面拷贝 `hello` 文件所在的目录，使用 `ls -l` 查看一下 `hello` 文件的权限；
- (2) 执行 `chmod 777 hello` 命令更改 `hello` 的权限，使之具备可执行属性；
- (3) 在命令提示符下输入 `./hello` 并回车，即可执行该程序。

实验六 Bootloader 定制实验（可选）

一、实验目的

- 1、了解 Bootloader 的构成。
- 2、掌握 U-Boot 的编译及移植过程。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、在顶层 Makefile 中为实验所用 ARM 处理器添加新的配置选项。
步骤：
- 2、在 board 目录中创建一个实验箱的目录。
步骤：
- 3、为实验箱添加新的配置文件。
步骤：
- 4、选择板级配置，编译 U-Boot。
步骤：

实验七 嵌入式 Linux 内核的配置、裁减及编译

一、实验目的

- 1、了解 Linux 内核的裁减方法。
- 2、掌握嵌入式 Linux 内核的配置及编译过程。

二、实验设备

PC 机。

三、实验内容

- 1、下载一个 2.6.XX 版本的 Linux 内核源代码，并在虚拟机中进行解压。

步骤：（注：以下输入的字符中不包含双引号）

- （1）打开网站 <https://www.kernel.org/pub>，点击 linux/进入下一级目录，然后点击 kernel/进入下一级目录，再点击 v2.6/进入下一级目录，在这一级目录中有从 2.6.0 到 2.6.39.4 的内核源代码文件及部分补丁；
- （2）点击 linux-2.6.39.tar.bz2 文件进行下载，下载完成后就得到一个压缩包文件；
- （3）把下载得到的文件 linux-2.6.39.tar.bz2 上传到虚拟机的/share 目录下，在命令行下输入“tar -jxvf linux-2.6.39.tar.bz2”并回车进行解压；
- （4）解压完成后会生成一个 linux-2.6.39 的目录，进入该目录后就可以看到内核源代码的目录树了。

- 2、根据所选用的 ARM 处理器，选择一个配置模板文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在源码根目录下输入“cd arch/arm/configs”并回车进入配置模板文件目录；
- （2）在该目录下找到一个名为 s3c2410_defconfig 的配置文件，并把它拷贝到源码根目录下变更为 .config，输入命令“cp s3c2410_defconfig ../../../.config”并回车即可；

- 3、根据需要修改 makefile 文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）进入源码根目录，输入“vi Makefile”并回车打开 Makefile 文件；
- （2）找到其中的“ARCH ?=\$(SUBARCH)”一行（大约在第 195 行处），把它改为“ARCH ? = arm”；

- (3) 下行“CORSS_COMPILE ? = \$(CONFIG_CROSS_COMPILE:”%”=%)”
把它改为 “CORSS_COMPILE ? = arm-linux-” (-后面不能有空格)；
- (4) 完成后存盘退出。

4、利用 make menuconfig 对内核配置进行修改。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 进入源码根目录，输入命令“make menuconfig”并回车，稍等一会儿后会打开一个文本方式下的配置对话框，如图 7.4-1 所示；

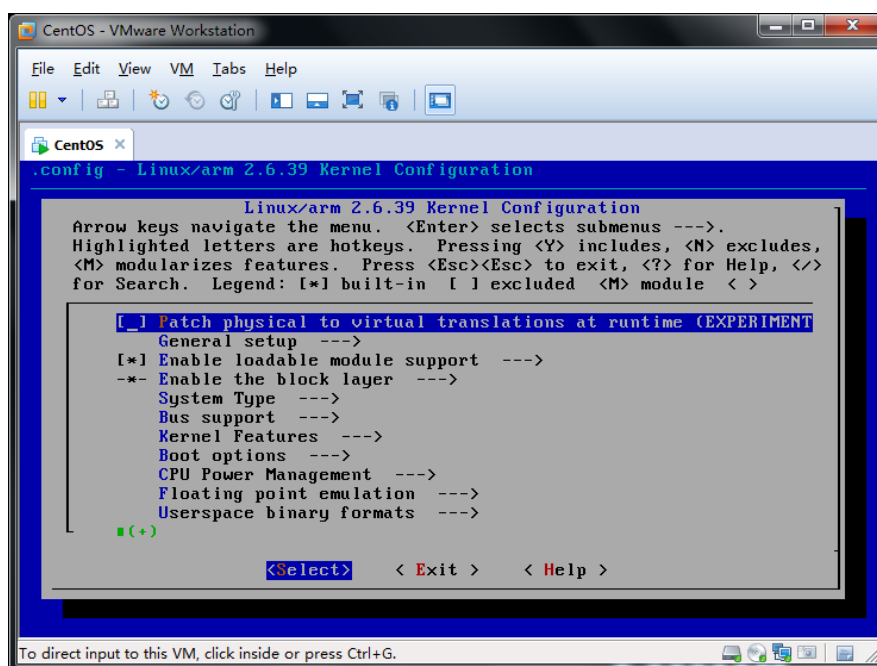


图 7.4-1

- (2) 在对话框中，利用上下光标键移动选择条，空格键勾选，回车键进入下一级设置，Tab 键在项目间跳转；
- (3) 本设置是以前面选择的 s3c2410 为基础的，一般需要在此基础上进行适当的裁减和调整以适应硬件环境，但本例仅仅是一个实验，所以可全部采用默认值而不做任何更改（有兴趣的可自行研究），这里直接按 Tab 键聚焦到 Exit 按钮上并回车，会问是否要保存配置，直接 yes 即可把配置保存到.config 文件中。

5、利用 make zImage 对内核进行编译压缩。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 进入源码根目录，输入命令“make zImage”并回车进行内核的编译，内核编译比较耗费时间，需要耐心等待；
- (2) 编译结束后，会在 arch/arm/boot 目录下形成内核文件 zImage，可输入“ls arch/arm/boot/zImage”进行查看，一般会有 2M 多一点；
- (3) 若要删除配置及生成的文件，有三个可以执行的命令：“make clean”、“make mrproper”和“make distclean”，第一个仅删除大部分基

本文件但保留配置文件（生成的 `zImage` 也会补删除），第二删除所有基本文件和配置文件，第三个删除所有生成的文件及其补丁文件，一般情况下用第一或第二个命令就可以了，一般要从头再来才会去执行第三个命令。

实验八 嵌入式根文件系统的制作

一、实验目的

- 1、了解嵌入式根文件系统的构成。
- 2、掌握嵌入式根文件系统的制作过程。

二、实验设备

PC 机。

三、实验内容

- 1、创建根文件系统目录，创建设备文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的/share 目录下新建一个名为 rootfs 的目录，输入命令“`mkdir /share/rootfs`”，在该目录下创建几个根文件系统必要的目录，输入“`mkdir bin sbin etc dev proc lib sys var mnt usr`”并回车；
- （2）执行“`cd dev`”进入该目录，执行命令“`mknod console c 5 1`”创建 console 设备节点，执行命令“`mknod null c 1 3`”创建 null 设备节点。

- 2、编译内核模块，安装内核模块。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机中进入实验箱所用 Linux 的内核源代码目录，输入命令“`cd /arm2410cl/kernel/linux-2.4.18-2410cl`”并回车，然后执行交叉编译内核模块的命令“`make modules ARCH=arm CORSS_COMPILE=armv4l-unknown-linux-`”；
- （2）编译成功后进行安装，输入“`make modules_install ARCH=arm INSTALL_MOD_PATH=/share/rootfs`”并回车即可。

- 3、配置、编译、安装 busybox。

步骤：（注：以下输入的字符中不包含双引号）

- （1）busybox 可从网上下载，本例直接使用实验箱配套光盘提供的版本，位于光盘目录“Linux\rootfs”下，把它拷贝到虚拟机的/share 目录下并解压，输入“`tar -jxvf busybox-1.00-pre10.tar.bz2`”回车即可；
- （2）进入该目录，执行“`cd busybox-1.00-pre10`”，执行拷贝更名命令“`cp config-uptech .config`”，以使用光盘提供的配置文件（也可执行 `make menuconfig` 进行自定义配置，具体请自行研究）；

- (3) 先执行“make”命令进行编译，完成后再进行安装；
- (4) 安装时要安装到 rootfs 目录下，执行“make PREFIX=/share/rootfs install”（注：在有些版本中 PREFIX 要写成 CONFIG_PREFIX 才行）即可。

4、安装库文件。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 进入到交叉编译工具所在的目录，这里以实验箱光盘上提供的交叉编译工具为例，输入“cd /opt/host/armv4l/armv4l-unknown-linux”并回车进入目录；
- (2) 拷贝该目录中 lib 目录下的所有文件到 rootfs 目录的 lib 目录下，拷贝时要注意，软链接文件要保持为链接形式，不要拷贝成原文件，执行命令“cp .lib/*.so* /share/rootfs/lib -d”即可。

5、配置自动挂载文件系统。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在 rootfs 的 etc 目录下新建一个名为 inittab 的文件，执行“vi /share/rootfs/etc/inittab”，在其中录入图 8.5-1 所示的内容并存盘退出；

```
console::askfirst:~/bin/sh
::sysinit:/etc/init.d/rcS
```

图 8.5-1

- (2) 在 rootfs 的 etc 目录下新建一个 init.d 的目录，执行“mkdir /share/rootfs/etc/init.d”，然后在该目录下创建一个名为 rcS 的文本文件，执行“vi /share/rootfs/etc/init.d/rcS”，在其中录入图 8.5-2 所示的内容并存盘退出，然后给该文件加入可执行属性，执行“chmod 777 rcS”；

```
mount -a
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

图 8.5-2

- (3) 在 rootfs 的 etc 目录下新建一个名为 fstab 的文件，执行“vi /share/rootfs/etc/fstab”，在其中录入如图 8.5-3 所示的内容并存盘退出。

```
proc    /proc    proc    defaults    0    0
sysfs   /sys      sysfs   defaults    0    0
tmpfs   /dev      tmpfs   defaults    0    0
```

图 8.5-3

6、制作 root.cramfs 镜像文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在光盘目录“Linux\rootfs”下拷贝 cramfs 镜像制作工具 mkfs.cramfs 到虚拟机的/share 目录下（mkfs.cramfs 文件也可自己制作，具体过程请自行研究）；
- （2）执行命令“./mkfs.cramfs rootfs ./ root.cramfs”，成功后就在/share 目录下生成了根文件镜像 root.cramfs。

实验九 嵌入式 Linux 下的进程控制实验

一、实验目的

- 1、了解 Linux 内核的进程控制方法。
- 2、掌握嵌入式 Linux 内核的进程控制过程。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、利用 `getpid` 函数获取进程 ID 号，利用 `getppid` 函数获取父进程 ID 号。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的 `/share` 目录下新建一个名为 `pid` 的目录，输入命令 “`mkdir /share/pid`”，在该目录下创建一个名为 `getpid.c` 的文件，输入命令 “`vi /share/pid/getpid.c`”；
- （2）在 `getpid.c` 中录入图 9.1-1 所示的内容并存盘退出；

```
#include <stdio.h>
int main(void)
{
    printf("PID = %d\n",getpid());
    printf("PPID = %d\n",getppid());
    return 0;
}
```

图 9.1-1

- （3）在 `pid` 目录下输入命令 “`gcc getpid.c -o getpid`” 并回车进行编译；
- （4）若生成的 `getpid` 不具备可执行属性则修改它，输入 “`chmod 777 getpid`” 回车即可；
- （5）执行程序，输入 “`./getpid`” 并回车，执行结果如图 9.1-2 所示；
- （6）以上结果中，打印出的本进程及父进程的 ID 号数值会因环境不一样而不相同，不一定是图中的数值。

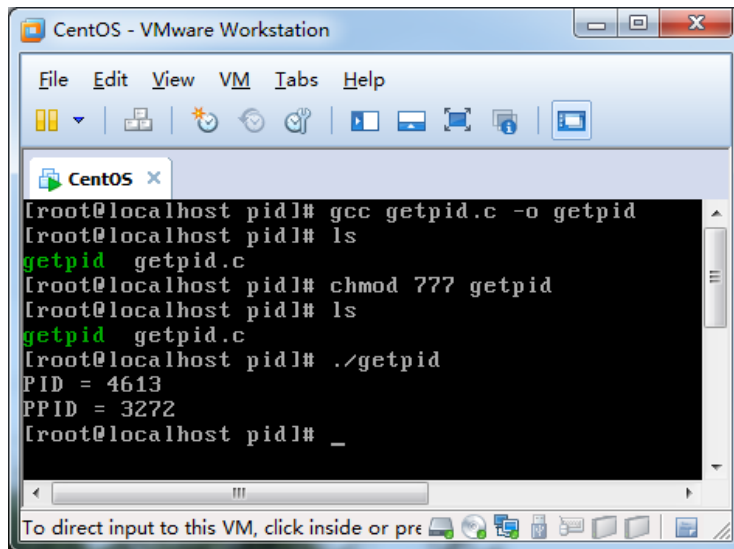


图 9.1-2

2、分别利用 fork、vfork 及 exec 函数族进行进程的创建，利用 wait 函数实现进程等待。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的/share 目录下新建一个名为 fork 的目录，输入命令“mkdir /share/fork”，在该目录下创建一个名为 fork1.c 的文件，输入命令“vi /share/fork/fork1.c”；
- （2）在 fork1.c 中录入图 9.2-1 所示的内容并存盘退出；

```

#include <sys/types.h>
#include <stdio.h>
int main(void)
{
    pid_t pid;
    pid = fork();
    if( pid < 0 )
        printf("error in fork!");
    else if( pid == 0 )
        printf("I am the child process, ID is %d\n",getpid());
    else
        printf("I am the parent process, ID is %d\n",getpid());
    return 0;
}

```

图 9.2-1

- （3）在 fork 目录下输入命令“gcc fork1.c -o fork1”并回车进行编译；
- （4）若生成的 fork1 不具备可执行属性则修改它，输入“chmod 777 fork1”回车即可；
- （5）执行程序，输入“./fork1”并回车，执行结果如图 9.2-2 所示。

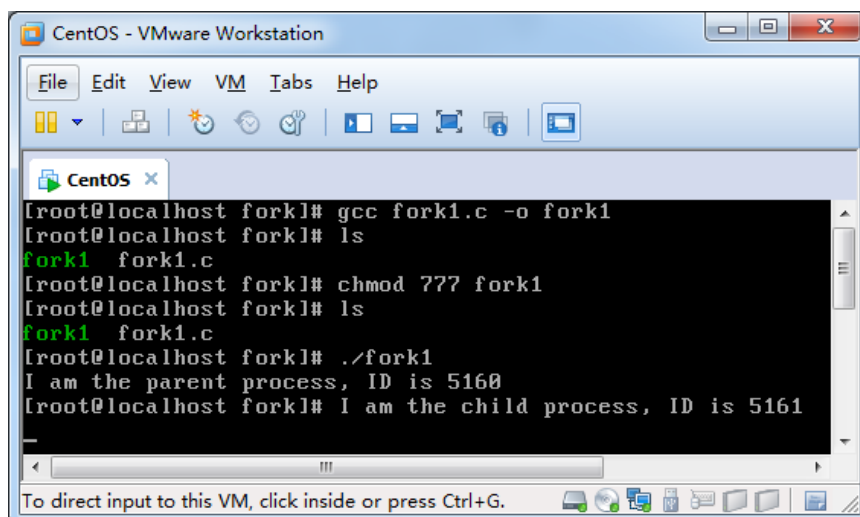


图 9.2-2

- (6) 在 fork 目录下创建一个名为 fork2.c 的文件，输入命令 “vi /share/fork/fork2.c”；
- (7) 在 fork2.c 中录入图 9.2-3 所示的内容并存盘退出；

```

#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
int main(void)
{
    pid_t pid;
    int count = 0;
    pid = fork();
    count++;
    printf("count = %d\n",count);
    return 0;
}
  
```

图 9.2-3

- (8) 在 fork 目录下输入命令 “gcc fork2.c -o fork2” 并回车进行编译；
- (9) 若生成的 fork2 不具备可执行属性则修改它，输入 “chmod 777 fork2” 回车即可；
- (10) 执行程序，输入 “./fork2” 并回车，执行结果如图 9.2-4 所示。
- (11) 在 fork 目录下创建一个名为 vfork.c 的文件，输入命令 “vi /share/fork/vfork.c”；
- (12) 在 vfork.c 中录入图 9.2-5 所示的内容并存盘退出；
- (13) 在 fork 目录下输入命令 “gcc vfork.c -o vfork” 并回车进行编译；
- (14) 若生成的 vfork 不具备可执行属性则修改它，输入 “chmod 777 vfork” 回车即可；
- (15) 执行程序，输入 “./vfork” 并回车，执行结果如图 9.2-6 所示。

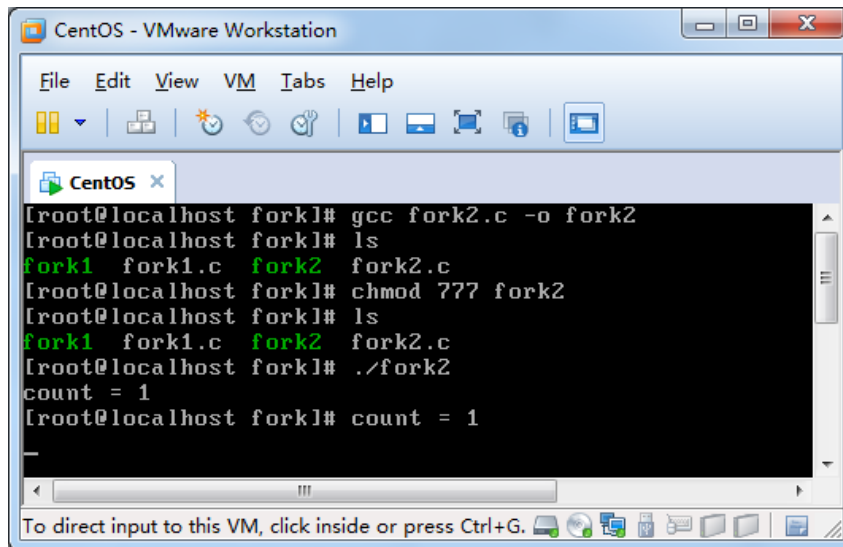


图 9.2-4

```

#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    pid_t pid;
    int count = 0;
    pid = vfork();
    count++;
    printf("count = %d\n",count);
    exit(0);
}

```

图 9.2-5

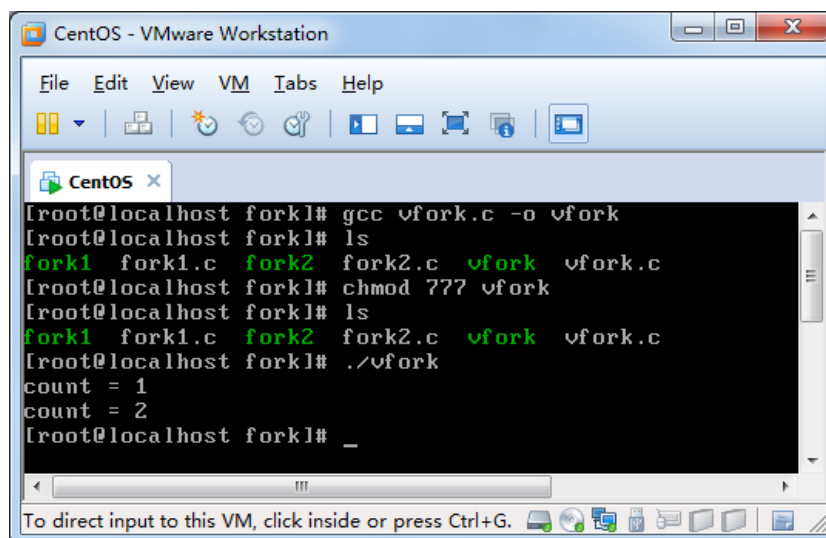


图 9.2-6

- (16) 在虚拟机的/share 目录下新建一个名为 exec 的目录，输入命令“mkdir /share/exec”，在该目录下创建一个名为 execl.c 的文件，输入命令“vi /share/exec/execl.c”；
- (17) 在 execl.c 中录入图 9.2-7 所示的内容并存盘退出；

```
#include <unistd.h>
int main(void)
{
    execl("/bin/ls", "ls", "-l", "/root/.bash_profile", (char *)0);
    return 0;
}
```

图 9.2-7

- (18) 在 exec 目录下输入命令“gcc execl.c -o execl”并回车进行编译；
- (19) 若生成的 execl 不具备可执行属性则修改它，输入“chmod 777 execl”回车即可；
- (20) 执行程序，输入“./execl”并回车，执行结果如图 9.2-8 所示。
- (21) 在 exec 目录下创建一个名为 execlp.c 的文件，输入命令“vi /share/exec/execlp.c”；
- (22) 在 execlp.c 中录入图 9.2-9 所示的内容并存盘退出；

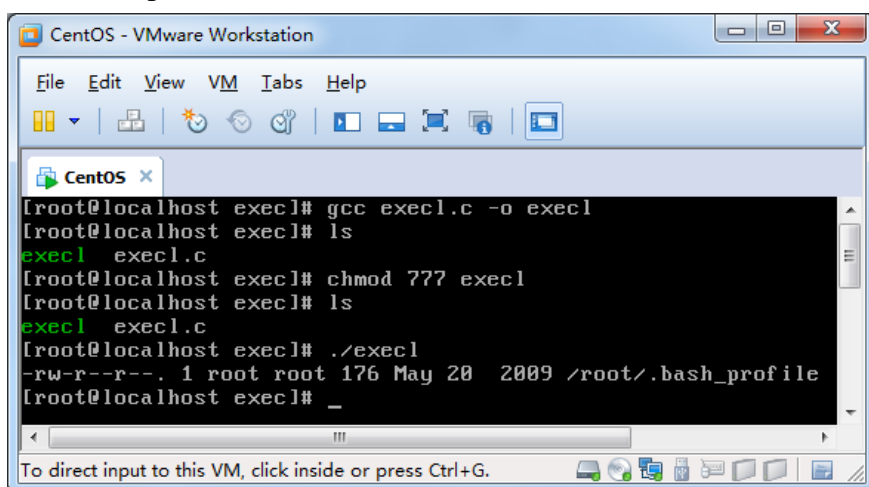


图 9.2-8

```
#include <unistd.h>
int main(void)
{
    execlp("ls", "ls", "-l", "/root/.bash_profile", (char *)0);
    return 0;
}
```

图 9.2-9

- (23) 在 exec 目录下输入命令“gcc execlp.c -o execlp”并回车进行编译；

- (24) 若生成的 `execlp` 不具备可执行属性则修改它，输入“`chmod 777 execlp`”回车即可；
- (25) 执行程序，输入“`./execlp`”并回车，执行结果如图 9.2-10 所示。

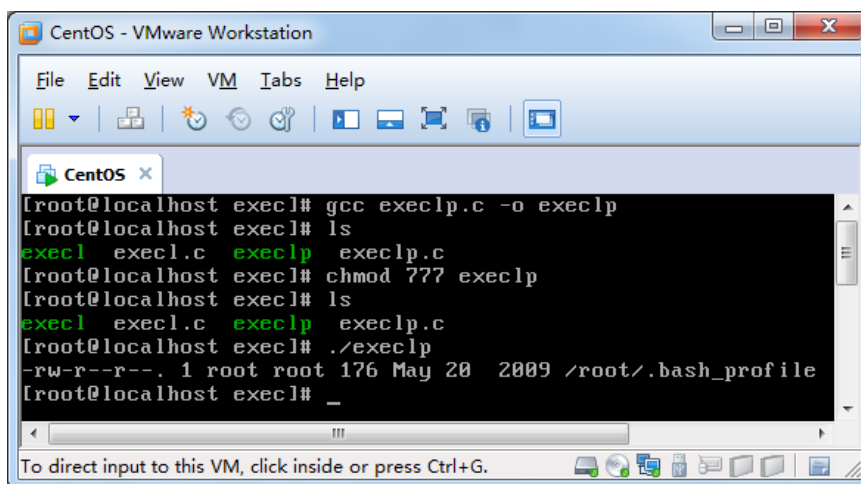


图 9.2-10

- (26) 在 `exec` 目录下创建一个名为 `execv.c` 的文件，输入命令“`vi /share/exec/execv.c`”；
- (27) 在 `execv.c` 中录入图 9.2-11 所示的内容并存盘退出；
- (28) 在 `exec` 目录下输入命令“`gcc execv.c -o execv`”并回车进行编译；

```
#include <unistd.h>
int main(void)
{
    char * argv[] = {"ls", "-l", "/root/.bash_profile", (char *)0};
    execv("/bin/ls", argv);
    return 0;
}
```

图 9.2-11

- (29) 若生成的 `execv` 不具备可执行属性则修改它，输入“`chmod 777 execv`”回车即可；
- (30) 执行程序，输入“`./execv`”并回车，执行结果如图 9.2-12 所示。
- (31) 在 `exec` 目录下创建一个名为 `system.c` 的文件，输入命令“`vi /share/exec/system.c`”；
- (32) 在 `system.c` 中录入图 9.2-13 所示的内容并存盘退出；
- (33) 在 `exec` 目录下输入命令“`gcc system.c -o system`”并回车进行编译；
- (34) 若生成的 `system` 不具备可执行属性则修改它，输入“`chmod 777 system`”回车即可；
- (35) 执行程序，输入“`./system`”并回车，执行结果如图 9.2-14 所示。

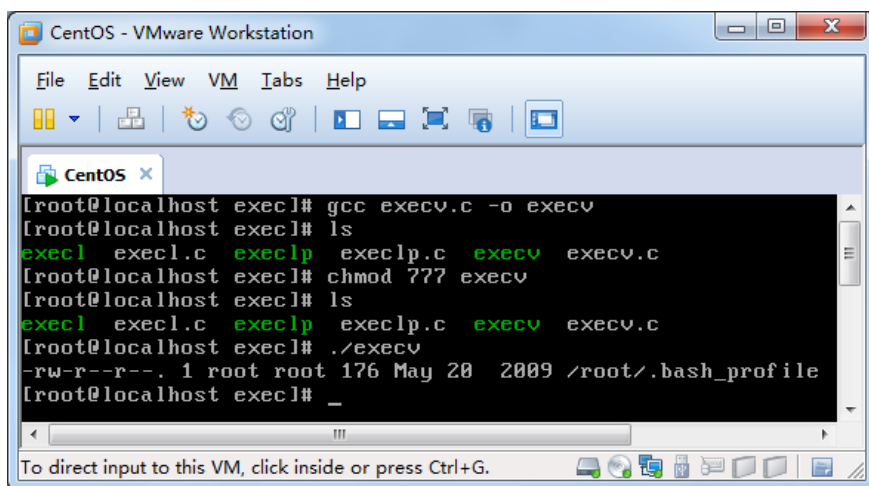


图 9.2-12

```

#include <stdio.h>
int main(void)
{
    system("ls -l /root/.bash_profile");
    return 0;
}

```

图 9.2-13

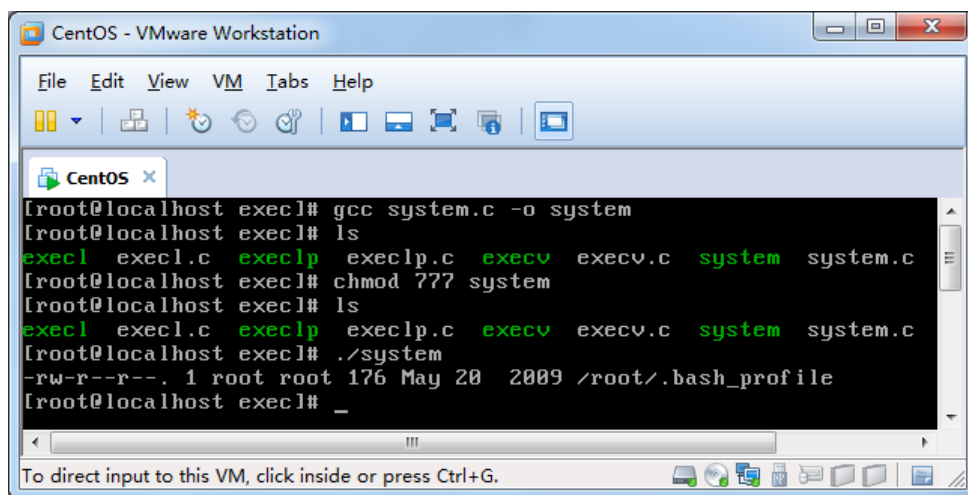


图 9.2-14

- (36) 在虚拟机的/share 目录下新建一个名为 wait 的目录，输入命令“mkdir /share/wait”，在该目录下创建一个名为 wait.c 的文件，输入命令“vi /share/wait/wait.c”；
- (37) 在 wait.c 中录入图 9.2-15 所示的内容并存盘退出；
- (38) 在 wait 目录下输入命令“gcc wait.c -o wait”并回车进行编译；
- (39) 若生成的 wait 不具备可执行属性则修改它，输入“chmod 777 wait”回车即可；
- (40) 执行程序，输入“./wait”并回车，执行结果如图 9.2-16 所示。


```

#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    pid_t pc,pr;
    pc = fork();
    if( pc == 0 )
    {
        printf("This is child process with pid of %d\n",getpid());
        sleep(10);
    }
    else if( pc > 0 )
    {
        pr = wait(NULL);
        printf("I caught a child process with pid of %d\n",pr);
    }
    exit(0);
    return 0;
}

```

图 9.2-15

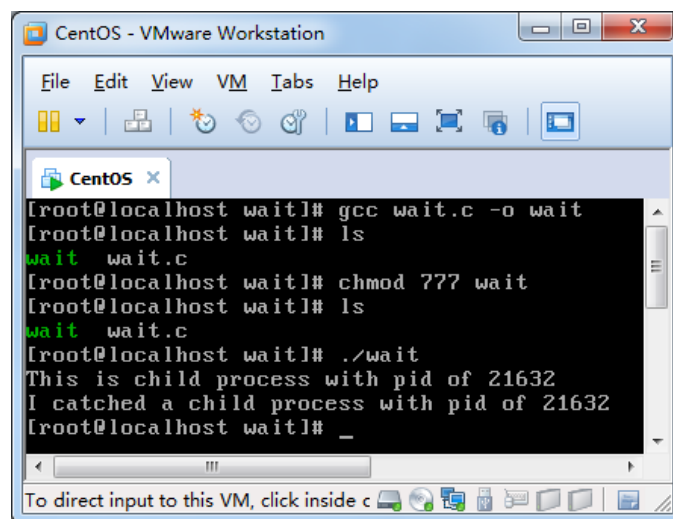


图 9.2-16

3、利用无名 pipe 及有名 pipe 实现进程间的通信。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的/share 目录下新建一个名为 pipe 的目录，输入命令“mkdir /share/pipe”，在该目录下创建一个名为 pipe.c 的文件，输入命令“vi /share/pipe/pipe.c”；
- （2）在 pipe.c 中录入图 9.3-1 所示的内容并存盘退出；

```

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int pipe_fd[2];
    pid_t pid;
    char buf_r[100];
    char* p_wbuf;
    int r_num;
    memset(buf_r,0,sizeof(buf_r));
    if(pipe(pipe_fd)<0)
    {
        printf("pipe create error\n");
        return -1;
    }
    if((pid=fork())==0)
    {
        printf("\n");
        close(pipe_fd[1]);
        sleep(2);
        if((r_num=read(pipe_fd[0],buf_r,100))>0)
            printf("%d numbers read from the pipe is %s\n",r_num,buf_r);
        close(pipe_fd[0]);
        exit(0);
    }
    else if(pid>0)
    {
        close(pipe_fd[0]);
        if(write(pipe_fd[1],"Hello",5) != -1)
            printf("parent write1 Hello!\n");
        if(write(pipe_fd[1]," Pipe",5) != -1)
            printf("parent write2 Pipe!\n");
        close(pipe_fd[1]);
        sleep(3);
        waitpid(pid,NULL,0);
        exit(0);
    }
    return 0;
}

```

图 9.3-1

- (3) 在 pipe 目录下输入命令 “gcc pipe.c -o pipe” 并回车进行编译;
- (4) 若生成的 pipe 不具备可执行属性则修改它, 输入 “chmod 777 pipe” 回车即可;

- (5) 执行程序，输入“./pipe”并回车，执行结果如图 9.3-2 所示。

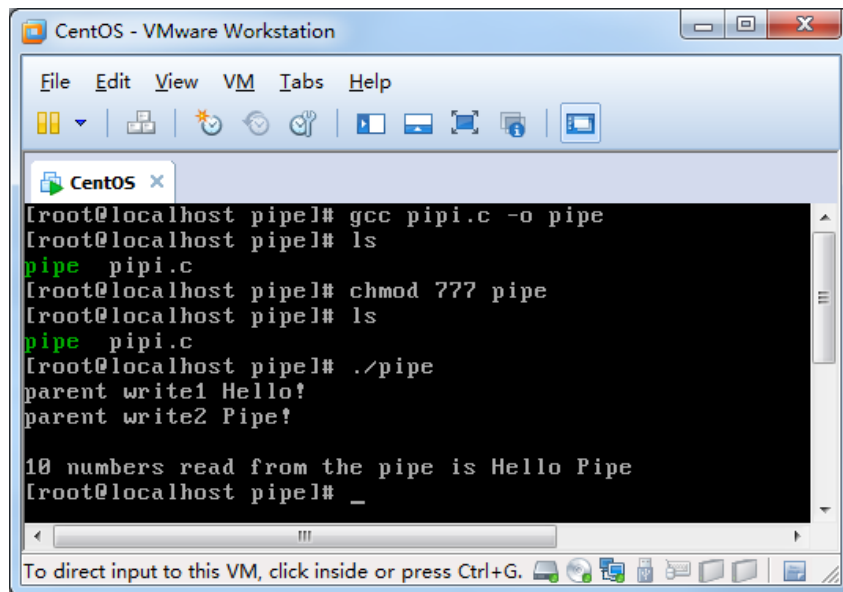


图 9.3-2

- (6) 在 pipe 目录下创建一个名为 fifo_write.c 的文件，输入命令“vi /share/pipe/fifo_write.c”；
- (7) 在 fifo_write.c 中录入图 9.3-3 所示的内容并存盘退出；

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FIFO_SERVER "/tmp/myfifo"
int main(int argc, char** argv)
{
    int fd;
    char w_buf[100];
    int nwrite;
    fd=open(FIFO_SERVER,O_WRONLY|O_NONBLOCK,0);
    if(argc==1)
    {
        printf("Please send something\n");
        exit(-1);
    }
    strcpy(w_buf,argv[1]);
    if((nwrite=write(fd,w_buf,100))==-1)
    {
        if(errno==EAGAIN)
            printf("The FIFO has not been read yet.Please
try later\n");
    }
}
```

```

else
    printf("write %s to the FIFO\n",w_buf);
}

```

图 9.3-3

- (8) 在 pipe 目录下输入命令 “gcc fifo_write.c -o fifo_write” 并回车进行编译;
- (9) 若生成的 fifo_write 不具备可执行属性则修改它, 输入 “chmod 777 fifo_write” 回车即可;
- (10) 先不运行它, 再次在 pipe 目录下创建一个名为 fifo_read.c 的文件, 输入命令 “vi /share/pipe/fifo_read.c” ;
- (11) 在 fifo_read.c 中录入图 9.3-4 所示的内容并存盘退出;
- (12) 在 pipe 目录下输入命令 “gcc fifo_read.c -o fifo_read” 并回车进行编译;
- (13) 若生成的 fifo_read 不具备可执行属性则修改它, 输入 “chmod 777 fifo_read” 回车即可;
- (14) 由于是两个进程间的通信, 所以还要打开一个终端以便于测试操作, 这里使用 SSH 方式连接到虚拟机, 可通过 SecureCRT 打开一个终端窗口;

```

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FIFO "/tmp/myfifo"
int main(int argc,char** argv)
{
    char buf_r[100];
    int fd;
    int nread;
    if((mkfifo(FIFO,O_CREAT|O_EXCL)<0)&&(errno!=EEXIST))
        printf("cannot create fifoserver\n");
    printf("Preparing for reading bytes...\n");
    memset(buf_r,0,sizeof(buf_r));
    fd=open(FIFO,O_RDONLY|O_NONBLOCK,0);
    if(fd==-1)
    {
        perror("open");
        exit(1);
    }
    while(1)

```

```

{
    memset(buf_r,0,sizeof(buf_r));
    if((nread=read(fd,buf_r,100))==-1)
    {
        if(errno==EAGAIN)
            printf("no data yet\n");
    }
    printf("read %s from FIFO\n",buf_r);
    sleep(1);
}
pause();
unlink(FIFO);
}

```

图 9.3-4

- (15) 首先在虚拟机的 /share/pipe 目录下运行 fifo_read 文件,输入“./fifo_read”并回车即可,此时会连续显示出“read from FIFO”的信息;
- (16) 然后在 SecureCRT 打开的终端窗口中,进入到 /share/pipe 目录下,输入“./fifo_write hello”并回车运行,此时可以在虚拟机中看到打印了一行“read hello from FIFO”,说明两个进程间通过有名管道通信成功,具体如图 9.3-5 和图 9.3-6 所示。

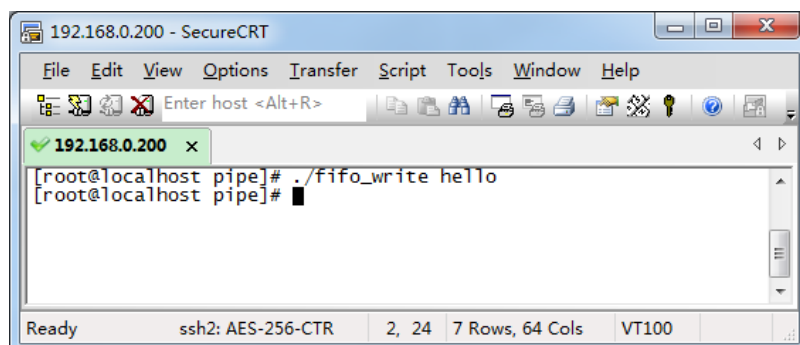


图 9.3-5

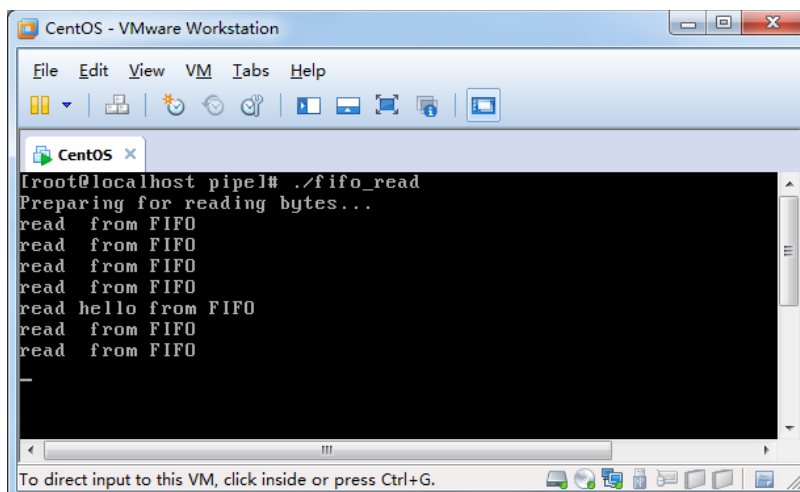


图 9.3-6

4、所有程序先在虚拟机上运行，然后交叉编译到嵌入式系统实验箱中运行。

步骤：（详略）

- （1）把前面所有实验中编译时使用的 `gcc` 换成 `arm-linux-gcc`（或 `armv4l-unknown-linux-gcc`）即可实现程序的交叉编译；
- （2）交叉编译后生成的文件通过网络输送到实验箱中，在实验箱中更改文件的可执行属性就可在实验箱中运行了，其余步骤不变。

实验十 嵌入式 Linux 下的文件编程实验

一、实验目的

- 1、了解 Linux 下文件编程的两种方式。
- 2、掌握系统调用方式进行文件编程的过程，为驱动开发做准备。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、利用系统调用函数 `creat` 实现文件的创建。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的 `/share` 目录下新建一个名为 `file` 的目录，输入命令 “`mkdir /share/file`”，在该目录下创建一个名为 `file_create.c` 的文件，输入命令 “`vi /share/file/file_create.c`”；
- （2）在 `file_create.c` 中录入图 10.1-1 所示的内容并存盘退出；

```
#include <stdio.h>
#include <stdlib.h>
void create_file(char *filename)
{
    if(creat(filename,0755)<0)
    {
        printf("create file %s failure!\n",filename);
        exit(EXIT_FAILURE);
    }
    else
        printf("create file %s success!\n",filename);
}
int main(int argc,char *argv[])
{
    int i;
    if(argc<2)
    {
        perror("you haven't input the filename,please try again!\n");
        exit(EXIT_FAILURE);
    }
    else
    {
```

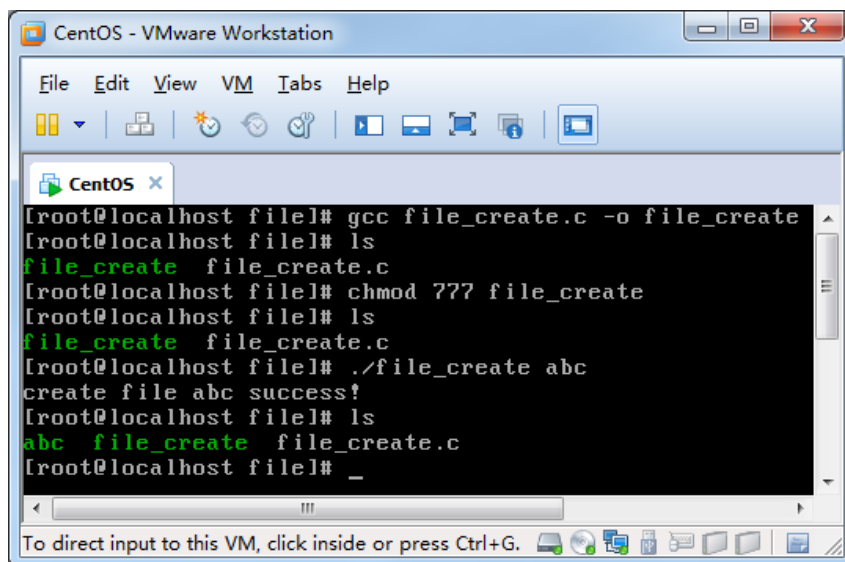
```

        for(i=1;i<argc;i++)
            create_file(argv[i]);
        exit(EXIT_SUCCESS);
    }
}

```

图 10.1-1

- (3) 在 file 目录下输入命令 “gcc file_create.c -o file_create” 并回车进行编译；
- (4) 若生成的 file_create 不具备可执行属性则修改它，输入 “chmod 777 file_create” 回车即可；
- (5) 执行程序，输入 “./file_create abc” 并回车，成功后可看见当前目录下新创建的文件 abc，执行结果如图 10.1-2 所示。



```

CentOS - VMware Workstation
File Edit View VM Tabs Help
CentOS x
[root@localhost file]# gcc file_create.c -o file_create
[root@localhost file]# ls
file_create file_create.c
[root@localhost file]# chmod 777 file_create
[root@localhost file]# ls
file_create file_create.c
[root@localhost file]# ./file_create abc
create file abc success!
[root@localhost file]# ls
abc file_create file_create.c
[root@localhost file]# _

```

图 10.1-2

2、利用系统调用函数 open 实现文件的打开及创建，close 实现对文件的关闭。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在 file 目录下创建一个名为 file_open.c 的文件，输入命令 “vi /share/file/file_open.c” ；
- (2) 在 file_open.c 中录入图 10.2-1 所示的内容并存盘退出；

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
int main(int argc,char *argv[])
{
    int fd;
    if(argc<2)

```



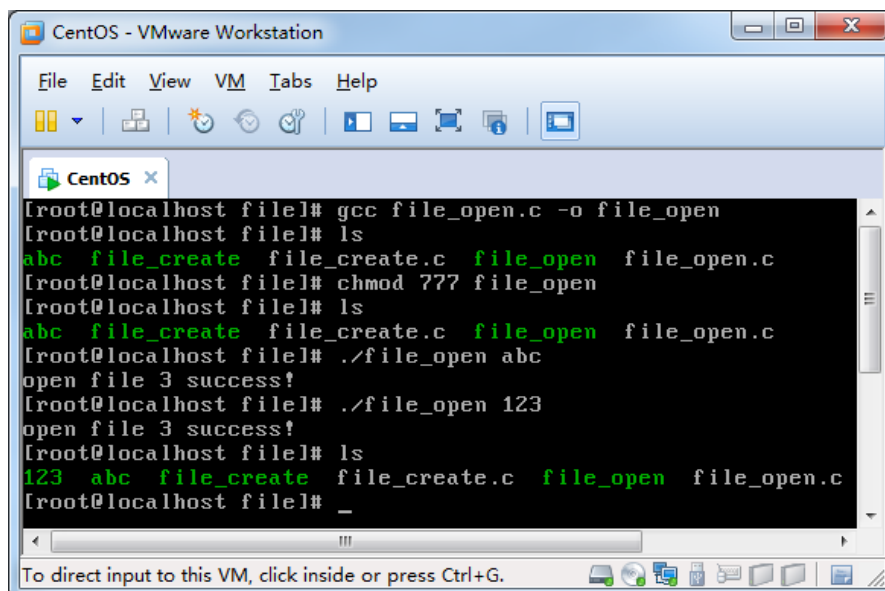
```

{
    puts("please input the open file pathname!\n");
    exit(1);
}
if((fd=open(argv[1],O_CREAT|O_RDWR,0777))<0)
{
    perror("open file failure!\n");
    exit(1);
}
else
    printf("open file %d success!\n",fd);
close(fd);
exit(0);
}

```

图 10.2-1

- (3) 在 file 目录下输入命令“gcc file_open.c -o file_open”并回车进行编译；
- (4) 若生成的 file_open 不具备可执行属性则修改它，输入“chmod 777 file_open”回车即可；
- (5) 执行程序，输入“./file_open abc”并回车，显示打开文件成功并返回文件描述符，接着输入“./file_open 123”并回车打开一个不存在的文件，可看见程序先创建了文件 123 然后再把它打开，同样成功的返回了文件描述符，执行结果如图 10.2-2 所示。



```

CentOS - VMware Workstation
File Edit View VM Tabs Help
CentOS x
[root@localhost file]# gcc file_open.c -o file_open
[root@localhost file]# ls
abc file_create file_create.c file_open file_open.c
[root@localhost file]# chmod 777 file_open
[root@localhost file]# ls
abc file_create file_create.c file_open file_open.c
[root@localhost file]# ./file_open abc
open file 3 success!
[root@localhost file]# ./file_open 123
open file 3 success!
[root@localhost file]# ls
123 abc file_create file_create.c file_open file_open.c
[root@localhost file]# _
To direct input to this VM, click inside or press Ctrl+G.

```

图 10.2-2

3、利用系统调用函数 read 及 write 实现对文件的读写。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在 file 目录下创建一个名为 file_copy.c 的文件，输入命令 “vi /share/file/file_copy.c”；
- (2) 在 file_copy.c 中录入图 10.3-1 所示的内容并存盘退出；

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#define BUFFER_SIZE 1024
int main(int argc,char **argv)
{
    int from_fd,to_fd;
    int bytes_read,bytes_write;
    char buffer[BUFFER_SIZE];
    char *ptr;
    if(argc != 3)
    {
        fprintf(stderr,"Usage:%s fromfile tofile/n/a",argv[0]);
        exit(1);
    }
    if((from_fd = open(argv[1],O_RDONLY)) == -1)
    {
        fprintf(stderr,"Open %s Error %s/n",argv[1],strerror(errno));
        exit(1);
    }
    if((to_fd = open(argv[2],O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR)) == -1)
    {
        fprintf(stderr,"Open %s Error %s/n",argv[2],strerror(errno));
        exit(1);
    }
    while(bytes_read = read(from_fd,buffer,BUFFER_SIZE))
    {
        if((bytes_read == -1) && (errno != EINTR)) break;
        else if(bytes_read > 0)
        {
            ptr = buffer;
            while(bytes_write = write(to_fd,ptr,bytes_read))
            {
                if((bytes_write == -1) && (errno != EINTR)) break;
                else if(bytes_write == bytes_read) break;
                else if(bytes_write > 0)
                {
                    ptr += bytes_write;
                    bytes_read -= bytes_write;
                }
            }
        }
    }
}
```

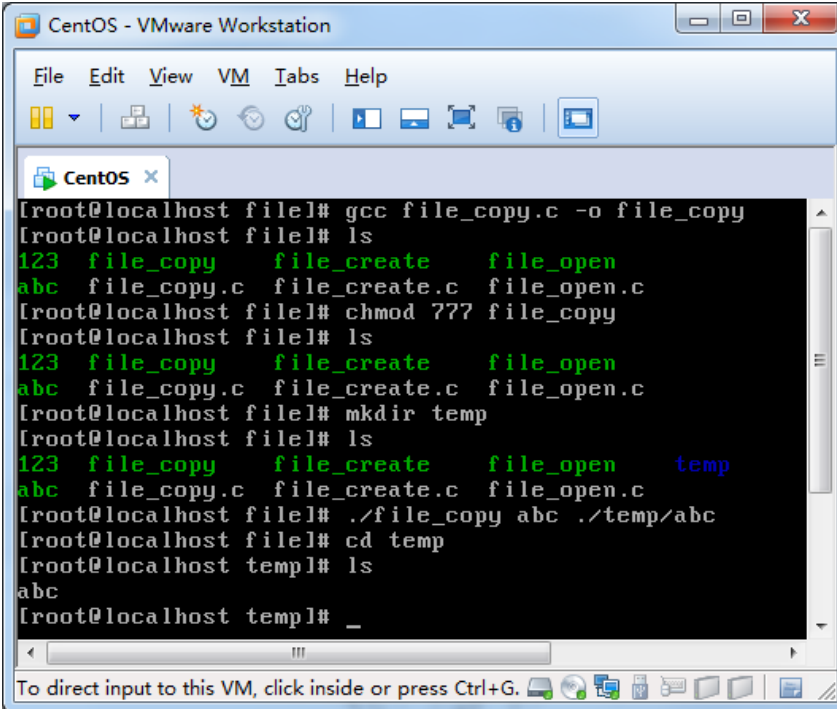
```

    }
    }
    if(bytes_write == -1) break;
}
}
close(from_fd);
close(to_fd);
exit(0);
}

```

图 10.3-1

- (3) 在 file 目录下输入命令“gcc file_copy.c -o file_copy”并回车进行编译；
- (4) 若生成的 file_copy 不具备可执行属性则修改它，输入“chmod 777 file_copy”回车即可；
- (5) 先在 file 目录下新建一个名为 temp 的目录，然后执行程序，输入“./file_copy abc ./temp/abc”并回车，然后进入到 temp 目录下，可看见文件 abc 已经被拷贝过来了，执行结果如图 10.3-2 所示。



```

CentOS - VMware Workstation
File Edit View VM Tabs Help
CentOS x
[root@localhost file]# gcc file_copy.c -o file_copy
[root@localhost file]# ls
123 file_copy file_create file_open
abc file_copy.c file_create.c file_open.c
[root@localhost file]# chmod 777 file_copy
[root@localhost file]# ls
123 file_copy file_create file_open
abc file_copy.c file_create.c file_open.c
[root@localhost file]# mkdir temp
[root@localhost file]# ls
123 file_copy file_create file_open temp
abc file_copy.c file_create.c file_open.c
[root@localhost file]# ./file_copy abc ./temp/abc
[root@localhost file]# cd temp
[root@localhost temp]# ls
abc
[root@localhost temp]# _

```

图 10.3-2

4、利用系统调用函数 lseek 实现对文件大小的测量。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在虚拟机的/share 目录下新建一个名为 lseek 的目录，输入命令“mkdir /share/lseek”，在该目录下创建一个名为 lseek.c 的文件，输入命令“vi /share/lseek/lseek.c”；

- (2) 在 lseek.c 中录入图 10.4-1 所示的内容并存盘退出;

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
int main(int argc,char *argv[])
{
    int fd;
    off_t currpos;
    if(argc<2)
    {
        puts("please input the open file pathname!\n");
        exit(1);
    }
    if((fd=open(argv[1],O_CREAT|O_RDWR,0777))<0)
    {
        perror("open file failure!\n");
        exit(1);
    }
    else
    {
        currpos = lseek(fd, 0, SEEK_END);
        printf("file %s size is : %ld\n",argv[1],currpos);
    }
    close(fd);
    exit(0);
}
```

图 10.4-1

- (3) 在 lseek 目录下输入命令“gcc lseek.c -o lseek”并回车进行编译;
- (4) 若生成的 lseek 不具备可执行属性则修改它,输入“chmod 777 lseek”回车即可;
- (5) 先在 lseek 目录下新建一个名为 hello 的文本文件,并在该文件中输入一句“Hello world!”,输入“echo Hello world! >> hello”并回车即可,然后输入“./lseek hello”并回车执行程序,此时可看到屏幕打印出文件 hello 的长度,结果如图 10.4-2 所示。

5、利用系统调用函数 access 实现对文件属性的判断。

步骤: (注: 以下输入的字符中不包含双引号)

- (1) 在虚拟机的/share 目录下新建一个名为 access 的目录,输入命令“mkdir /share/access”,在该目录下创建一个名为 access.c 的文件,输入命令“vi /share/access/access.c”;
- (2) 在 access.c 中录入图 10.5-1 所示的内容并存盘退出;

- (3) 在 access 目录下输入命令“gcc access.c -o access”并回车进行编译;

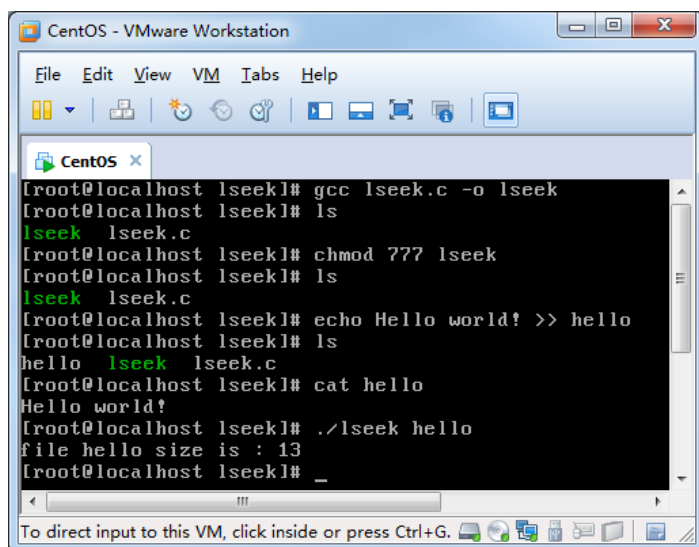


图 10.4-2

```
#include <unistd.h>
#include <stdio.h>
int main(void)
{
    if(access("/etc/passwd",R_OK) == 0)
        printf("/etc/passwd can be read!\n");
    else
        printf("/etc/passwd can not be read!\n");
}
```

图 10.5-1

- (4) 若生成的 access 不具备可执行属性则修改它，输入“chmod 777 access”回车即可;
- (5) 执行程序，输入“./access”并回车，可看到返回的/etc/passwd 文件属性可读，如图 10.5-2 所示。

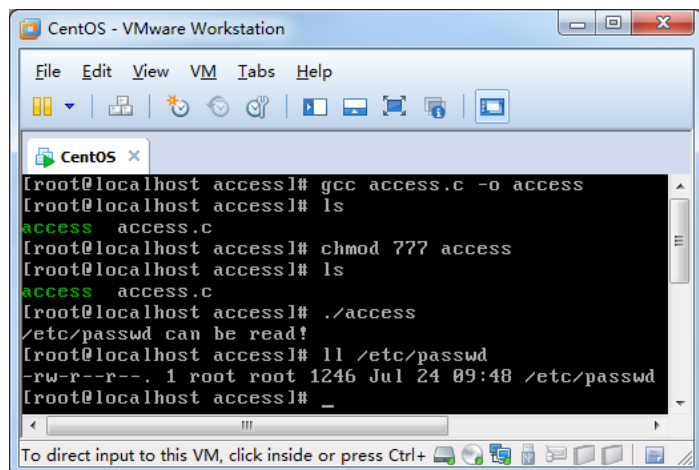


图 10.5-2

6、所有程序先在虚拟机上运行，然后交叉编译到嵌入式系统实验箱中运行。

步骤：（详略）

- （1）把前面所有实验中编译时使用的 `gcc` 换成 `arm-linux-gcc`（或 `armv4l-unknown-linux-gcc`）即可实现程序的交叉编译；
- （2）交叉编译后生成的文件通过网络输送到实验箱中，在实验箱中更改文件的可执行属性就可在实验箱中运行了，其余步骤不变。

实验十一 嵌入式 Linux 内核模块开发实验

一、实验目的

- 1、掌握内核模块的开发及使用方法，为驱动开发做准备。
- 2、掌握内核模块操作的相关命令。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、编写内核模块程序 `hello.c`，并编写配套的 `Makefile` 文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的 `/share` 目录下新建一个名为 `module` 的目录，输入命令“`mkdir /share/module`”，在该目录下创建一个名为 `hello.c` 的文件，输入命令“`vi /share/module/hello.c`”；
- （2）在 `hello.c` 中录入图 11.1-1 所示的内容并存盘退出；

```
#include <linux/module.h>
#include <linux/init.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("fengxun");
MODULE_DESCRIPTION("Hello World Module");
MODULE_ALIAS("a simplest module");
static int __init hello_init()
{
    printk(KERN_EMERG"Hello World!\n");
    return 0;
}
static void __exit hello_exit()
{
    printk("<1>Goodbye!\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

图 11.1-1

- （3）在 `module` 目录下创建一个名为 `Makefile` 的文件，输入命令“`vi /share/module/Makefile`”；
- （4）在 `Makefile` 中录入图 11.1-2 所示的内容并存盘退出；

```

ifneq ($(KERNELRELEASE),)
obj-m := hello.o
else
KDIR := /lib/modules/2.6.32-504.el6.i686/build
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    rm -f *.ko *.o *.mod.o *.mod.c *.symvers *.order
endif

```

图 11.1-2

2、执行 make 制作出内核模块 hello.ko。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在制作内核模块之前先要确保已安装了内核源代码，此处要特别注意，这里使用的内核源代码必须是当前运行的 Linux 版本的，而不是网上下载的，否则生成的内核模块不能插入到当前运行的 Linux 系统中去！如果系统是按照实验三中的项目二安装的，则内核源代码已经安装，进入目录“`cd /usr/src/kernels/2.6.32-504.el6.i686`”即可看到；
- （2）接下来输入“`cd /lib/modules/2.6.32-504.el6.i686`”并回车，查看一下内核模组目录，其中有两个链接文件 build 和 source，输入命令“`ls -l`”查看一下，source 是指向 build 的，而 build 则是指向 /usr/src/kernels/2.6.32-504.el6.i686 内核目录的，它们在编译内核模块时会用到；
- （3）在 module 目录下输入“make”并回车进行内核模块的编译，若成功则在当前目录会生成多个文件，其中名为 hello.o 和 hello.ko 的就是要用到的内核模块文件（一般 2.4 以下版本的内核使用 hello.o，2.6 以上版本的内核使用 hello.ko）；
- （4）输入命令“`modinfo hello.ko`”并回车查看一下模块信息。

3、把 hello.ko 模块插入内核，并观察屏幕打印信息。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在 module 目录下输入“`insmod hello.ko`”并回车，模块插入内核并在屏幕上打印出“Hello world!”的信息，如图 11.3-1 所示；
- （2）模块插入内核后，可通过执行命令“`lsmod`”来查看内核模块的情况。

4、把 hello 模块移出内核，并观察屏幕打印信息。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在 module 目录下输入“`rmmod hello`”（注意此处是 hello 没有 ko）并回车，模块移出内核并在屏幕上打印出“Goodbye!”的信息，如图 11.4-1 所示；

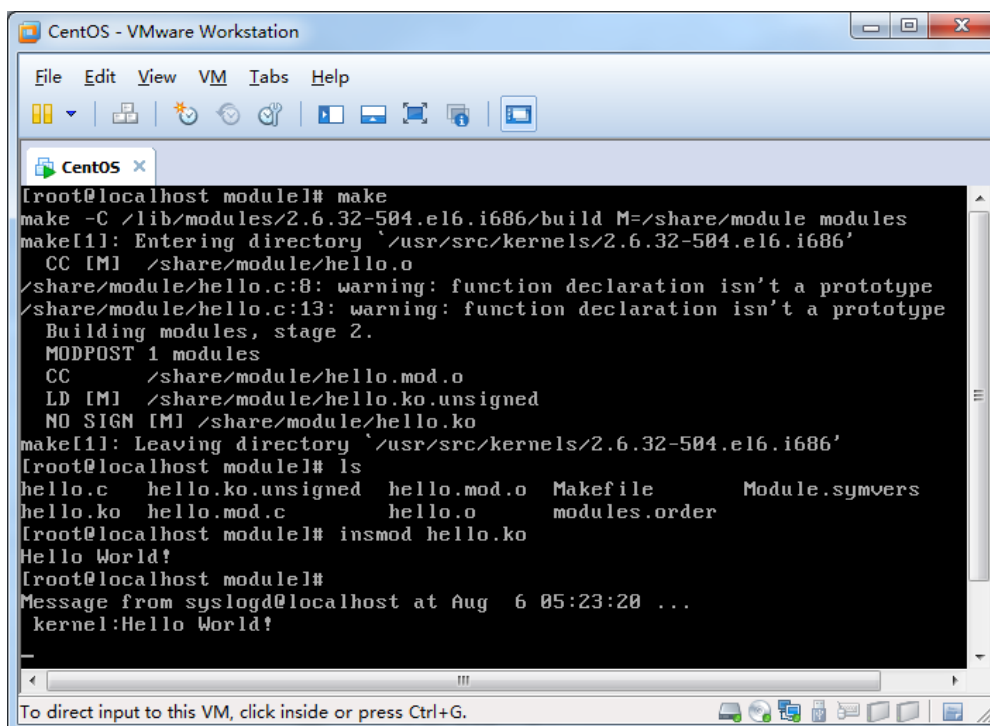


图 11.3-1

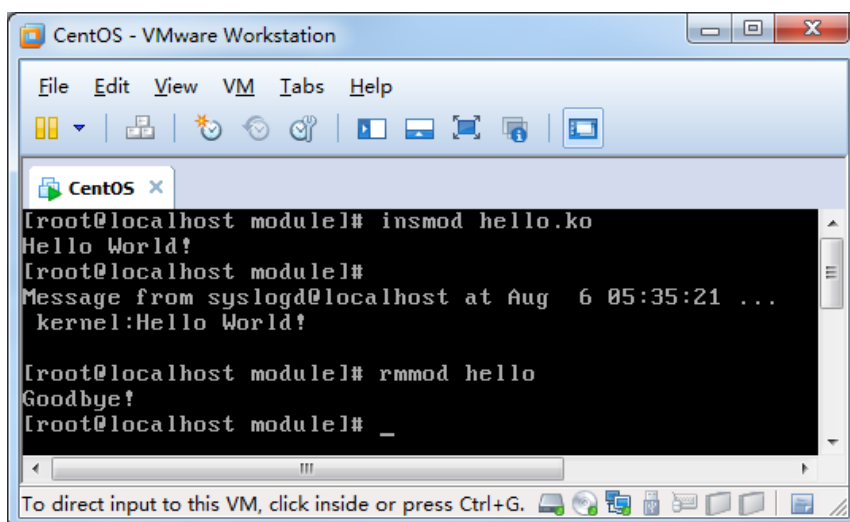


图 11.4-1

- (2) 模块移出内核后，再次通过命令“lsmod”可看到刚才插入的 hello 模块已经消失了。
- 5、所有程序先在虚拟机上运行，然后交叉编译到嵌入式系统实验箱中运行。

步骤：（注：此处实验箱上 Linux 的版本为 2.4 版）

- (1) 把前面实验中的 Makefile 文件内容换成图 11.5-1 的形式，hello.c 文件保持不变；

```

TOPDIR := .
KERNELDIR = /arm2410cl/kernel/linux-2.4.18-2410cl/
INCLUDEDIR = $(KERNELDIR)/include
CROSS_COMPILE=armv4l-unknown-linux-

AS      =$(CROSS_COMPILE)as
LD      =$(CROSS_COMPILE)ld
CC      =$(CROSS_COMPILE)gcc
CPP     =$(CC) -E
AR      =$(CROSS_COMPILE)ar
NM      =$(CROSS_COMPILE)nm
STRIP   =$(CROSS_COMPILE)strip
OBJCOPY =$(CROSS_COMPILE)objcopy
OBJDUMP =$(CROSS_COMPILE)objdump
CFLAGS += -I..
CFLAGS += -Wall -O -D__KERNEL__ -DMODULE -I$(INCLUDEDIR)

TARGET = hello.o
all: $(TARGET)
hello.o:hello.c
        $(CC) -c $(CFLAGS) $^ -o $@
clean:
        rm -f *.o *~ core .depend

```

图 11.5-1

- (2) 输入“make”并回车进行内核模块的交叉编译，若成功则在当前目录会生成一个名为 hello.o 的内核模块文件；
- (3) 把 hello.o 文件通过网络输送到实验箱中，进行插入和移出的操作，实验步骤不变。

实验十二 嵌入式 Linux 下的点灯（LED）实验

一、实验目的

- 1、掌握 Linux 下的驱动程序开发方法。
- 2、掌握字符型驱动的完整开发设计过程。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、查看实验箱的电路原理图，选定接有 LED 的端口，分析其亮/灭的电平状态。

步骤：（注：此处以“博创嵌入式系统实验箱”为例）

- （1）打开实验箱配套的光盘，进入目录“经典开发平台硬件文档\经典平台原理图\底板”，找到一个名为 Device.Sch 的 PDF 文件并打开它，在其中找到三个发光二极管的部分，如图 12.1-1 所示；

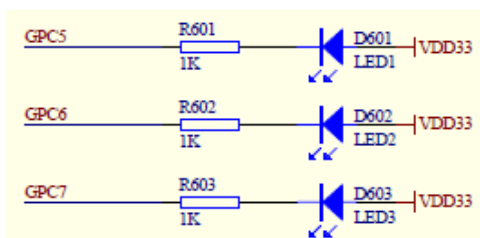


图 12.1-1

- （2）观察上图中的发光二极管接法，三个二极管为共阳接法，因此在给 s3c2410 的 GPC5、GPC6、GPC7 端口输出高电平时，三个发光管匀熄灭，输出低电平时，三个发光管匀点亮。
- 2、编写驱动模块程序 led_drv.c,同时编写一个配套的 Makefile 文件。

步骤：（注：以下输入的字符中不包含双引号）

- （1）在虚拟机的/share 目录下新建一个名为 led 的目录，输入命令“mkdir /share/led”，在该目录下创建一个名为 led_drv.c 的文件，输入命令“vi /share/led/led_drv.c”；
- （2）在 led_drv.c 中录入图 12.2-1 所示的内容并存盘退出；
- （3）在 led 目录下创建一个名为 Makefile 的文件，输入命令“vi /share/led/Makefile”；

```

#include <linux/module.h>
#include <linux/poll.h>
#include <asm/io.h>
#include <asm/hardware.h>
MODULE_LICENSE("GPL");
volatile unsigned long *gpcccon = NULL;
volatile unsigned long *gpcdat = NULL;
int major;
static int first_dev_open(struct inode *inode, struct file *file)
{
    *gpcccon &= ~((0x3<<(5*2)) | (0x3<<(6*2)) | (0x3<<(7*2)));
    *gpcccon |= ((0x1<<(5*2)) | (0x1<<(6*2)) | (0x1<<(7*2)));
    return 0;
}
static ssize_t first_dev_write(struct file *filp, const char *buf,
size_t count, loff_t * f_pos)
{
    int val = 1;
    copy_from_user(&val, buf, count);
    if (val == 1)
        *gpcdat &= ~((1<<5) | (1<<6) | (1<<7));
    else
        *gpcdat |= (1<<5) | (1<<6) | (1<<7);
    return 0;
}
static struct file_operations first_dev_fops = {
    .owner = THIS_MODULE,
    .open = first_dev_open,
    .write = first_dev_write,
};
static int first_dev_init(void)
{
    major = register_chrdev(0, "first_drv", &first_dev_fops);
    gpcccon = (volatile unsigned long *)ioremap(0x56000020, 16);
    gpcdat = gpcccon + 1;
    return 0;
}
static void first_dev_exit(void)
{
    unregister_chrdev(major, "first_drv");
}
module_init(first_dev_init);
module_exit(first_dev_exit);

```

图 12.2-1

(4) 在 Makefile 中录入图 12.2-2 所示的内容并存盘退出；

```
TOPDIR := .
KERNELDIR = /arm2410cl/kernel/linux-2.4.18-2410cl/
INCLUDEDIR = $(KERNELDIR)/include
CROSS_COMPILE=armv4l-unknown-linux-

AS      =$(CROSS_COMPILE)as
LD      =$(CROSS_COMPILE)ld
CC      =$(CROSS_COMPILE)gcc
CPP     =$(CC) -E
AR      =$(CROSS_COMPILE)ar
NM      =$(CROSS_COMPILE)nm
STRIP   =$(CROSS_COMPILE)strip
OBJCOPY =$(CROSS_COMPILE)objcopy
OBJDUMP =$(CROSS_COMPILE)objdump
CFLAGS += -I.
CFLAGS += -Wall -O -D__KERNEL__ -DMODULE -I$(INCLUDEDIR)

TARGET = led_dev.o
all: $(TARGET)
led_dev.o:led_dev.c
    $(CC) -c $(CFLAGS) $^ -o $@
clean:
    rm -f *.o *~ core .depend
```

图 12.2-2

3、执行 make 制作出驱动模块 led_drv.o。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在交叉编译内核驱动模块之前先要确保已有目标机的内核源代码（即实验箱上 Linux 的内核源码），本例使用“博创嵌入式系统实验箱”配套光盘上的内核源码，按光盘内手册上的要求，把内核源代码解压到目录“/arm2410cl/kernel/linux-2.4.18-2410cl”下；
- (2) 接下来要确保已有交叉编译工具链，本例使用“博创嵌入式系统实验箱”配套光盘上的交叉编译工具，按光盘内手册上的要求，把交叉编译工具链解压到目录“/opt/host/armv4l”下，并把其下的 bin 目录添加入搜索路径中（在 ~/.bash_profile 文件中加入一句“export PATH=/opt/host/armv4l/bin:\$PATH”即可）；
- (3) 在 led 目录下输入“make”并回车进行驱动模块的编译，若成功则在当前目录下会生成一个名为“led_drv.o”的驱动模块文件。

4、利用 insmod led_drv.o 命令插入内核，并用 lsmod 命令进行确认。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 把 led_drv.o 文件通过网络输送到实验箱中，在实验箱中执行命令“insmod led_drv.o”并回车把驱动模块插入到内核中去；
- (2) 模块插入内核后，通过执行命令“lsmod”来确保驱动模块已经插入到内核。

5、查看/proc/devices 文件内容，记录为 led_drv 模块分配的主设备号。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在实验箱下输入命令“cat /proc/devices”并回车，会显示一个设备号列表；
- (2) 在列表中找到刚才插入的 led_drv 模块，并记录下为其分配的主设备号，如图 12.5-1 所示。

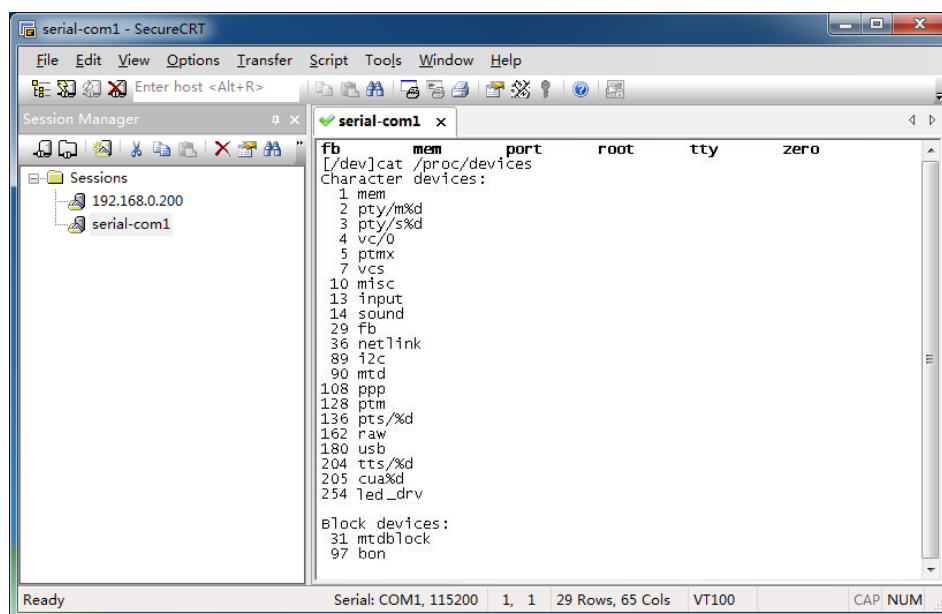


图 12.5-1

6、利用 mknod 命令建立一个名为 led 的字符设备节点文件。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在实验箱下输入命令“mknod /dev/led c 254 0”（注：此处的主设备号 254 要根据实际从图 12.5-1 中观察的主设备号来填写，不能照抄！）并回车，为插入的 led_drv 驱动模块建立一个设备节点；
- (2) 进入到设备目录确保设备文件 led 已经建立成功，输入命令“ls /dev/led”。

7、结合上述设备文件，编写相应的控制程序 led.c，并进行交叉编译生成应用程序 led，以利用其来控制 LED 的亮/灭。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在 led 目录下创建一个名为 led_test.c 的文件，输入命令“vi /share/led/led_test.c”；
- (2) 在 led_test.c 中录入图 12.7-1 所示的内容并存盘退出；

```

#include <stdio.h>
#include <fcntl.h>
int main(int argc, char **argv)
{
    int fd;
    int val = 1;
    fd = open("/dev/led", O_RDWR);
    if ( fd < 0 )
        printf("can`t open\n");
    if ( argc != 2 )
    {
        printf("Usage :\n");
        printf("%s <on|off>\n", argv[0]);
        return 0;
    }
    if ( strcmp(argv[1], "off") == 0 )
        val = 1;
    else
        val = 0;
    write(fd, &val, 4);
    return 0;
}

```

图 12.7-1

- (3) 在 led 目录下输入命令 “armv4l-unknown-linux-gcc led_test.c -o led_test” 并回车进行交叉编译；
- (4) 把 led_test 文件通过网络输送到实验箱中去，并在实验箱中执行命令 “chmod 777 led_test” 为其添加可执行属性。

8、本实验最终全部在嵌入式系统实验箱中运行。

步骤：（注：以下输入的字符中不包含双引号）

- (1) 在实验箱中执行命令 “./led_test on” 并回车，可看到实验箱上的三个 LED 均被点亮，如图 12.8-1 所示；
- (2) 在实验箱中执行命令 “./led_test off” 并回车，可看到实验箱上的三个 LED 均被熄灭，如图 12.8-2 所示。

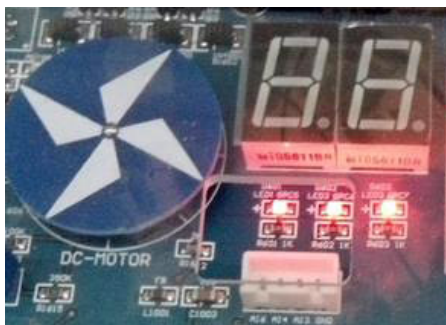


图 12.8-1

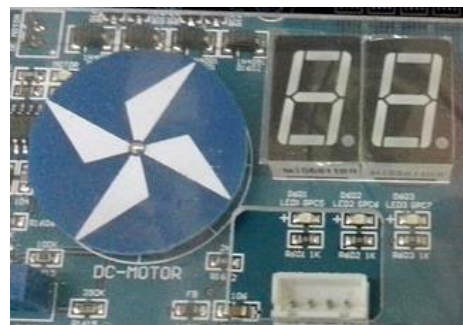


图 12.8-2

实验十三 嵌入式 Linux 下的 GUI 实验（可选）

一、实验目的

- 1、了解嵌入式 Linux 下的 GUI。
- 2、掌握相关 GUI 开发工具的使用。

二、实验设备

- 1、PC 机。
- 2、嵌入式系统实验箱。

三、实验内容

- 1、在 GUI 中设计一个按键，点击后弹出对话框显示“Hello World!”。
步骤：
- 2、在 QT 中进行程序设计，编译后先在 Linux 窗体中运行检查。
步骤：
- 3、交叉编译后下载到嵌入式系统实验箱中运行。
步骤：