

---

# 嵌入式程序设计实验手册

---

基于 EBD 嵌入式实验箱



2015-7-1

李龙海

# 实验一 嵌入式开发环境的搭建

## 一. 实验目的

通过本次实验，学生应掌握：

- 1. 交叉编译与交叉调试的概念。（重点）
- 2. 嵌入式 Linux 开发环境的搭建方法。
- 3. Linux 操作系统的使用方法，常用的 Linux 操作命令。
- 4. Linux 平台下 gcc 开发工具链的使用方法。（重点）
- 5. Linux 下基于源码部署和安装应用程序的方法。
- 6. （基于 C 语言）嵌入式 Linux 应用程序设计方法。（重点）

## 二. 预习内容：

### 1. 常用 Linux 命令

分类	命令名称	命令用途	重要程度
葵花宝典	man [command]	查看命令[command]的文档。	★★★★★
	info [command]	查看命令[command]的详细文档。	★★★
	[command] --help	查看命令[command]的简单帮助。	★★★★★
用户切换	su	变更用户身份。 无参数的默认情况提升为超级用户。	★★★★★
	sudo	以超级用户身份执行操作。	★★★★★
文件相关命令	ls	列出目录和文件信息。常用参数包括-l 和-a	★★★★★
	cd	改变当前工作目录。	★★★★★
	pwd	显示当前工作目录的绝对路径。	★★★
	cat	连接并显示指定的一个或多个文件的内容。	★★★
	cp	复制给出的文件或目录到另一文件或目录。	★★★★★

	mv	移动文件或目录到指定目录，或文件改名。	★★★★★
	rm	删除文件或目录。	★★★★★
	mkdir	创建目录。常用参数包括-p	★★★★★
	touch	创建一个新文件或修改文件访问时间。	★★
	chown	超级管理员用于修改文件的所有者和组别。	★★★★
	chgrp	超级管理员用于修改文件的组所有权。	★★★★
	chmod	改变文件的访问权限。	★★★★★
	ln	为某文件在另一个位置创建符号链接。	★★
	grep	在指定文件或目录中搜索特定的内容。	★★★★★
	find	在指定目录中搜索文件。	★★★★
	which	在 PATH 变量指定的路径中，搜索可执行文件的位置，并且返回第一个搜索结果。	★★
	whereis	在系统标准目录下搜索程序或文档。	★★
	file	查看文件类型。	★★★★
管道命令	>	输出重定向到文件。	★★★★
		输出重定向到程序。	★★★★
系统管理命令	ps	显示系统中由当前用户运行的进程列表。	★★★★★
	kill	输出特定的信号给指定的进程。	★★★★★
	top	动态显示系统中运行的程序相关信息。	★★★★★
	uname	显示系统的版本等信息。	★★★★
	shutdown	关闭或重启系统。	★★★★★
	reboot	重启或关闭系统。	★★★★★
	clear	清除屏幕上的信息。	★★★★★
磁盘相关命令	df	查看文件系统的磁盘空间占用情况。	★★★★
	du	统计目录或文件所占磁盘空间的大小。	★★
	free	查看当前系统内存使用情况。	★★
文件系统挂载命令	mount	挂载文件系统。 例如： mount -t vfat /dev/hdb1 /mnt/udisk	★★★★★
	umount	卸载文件系统。 例如： umount /mnt/udisk	★★★★★
环境变量相关命令	echo \$PATH	输出执行文件路径变量 PATH。	★★★★
	set	查看环境变量	
	export	定义或改变环境变量。 例如：export \$PATH=\$PATH:[path]	★★★★★
文件压缩相关命令	tar	对文件进行打包或者解包。 例如： tar xvzf hello.tar.gz tar jxvf hello.tar.bz2	★★★★★
	gzip	对文件进行压缩或解压缩。	★★★★

查看文件内容	more less	分页显示文件内容	★★★★
	head tail	显示文件开头/结尾的指定行数信息	★★★★
文件比较合并相关命令	diff	比较两个不同的文件或目录下的同名文件，生成补丁文件。 例如： diff -uN file1 file2 > patchfile	★★★★★
	patch	用补丁文件更新现有文件，与 diff 配合使用。 例如： patch -p0 file1 < patchfile	★★★★★
网络相关命令	ifconfig	查看和配置网络接口的地址和参数。	★★★★★★
	netstat	显示网络连接，路由表和网络接口信息。	★★★★
	ping	一般用于测试网络连通性，延迟丢包等状态。	★★★★★★

**提示：**如果执行某个命令“command”时出现权限相关的错误，可以尝试用命令“sudo command”重新执行一次（可能需要输入口令），即以管理员身份执行该命令。

2. gcc 编译器的用法
3. make 工具的用法、Makefile 文件的编写方法
4. gdb 调试工具的用法
5. vi 编辑工具的用法。

### 三. 实验内容

本次实验的主要实验内容包括：

1. 搭建宿主机工作环境
  - a) 在宿主机（物理机或虚拟机）上安装 Linux 操作系统。
  - b) 在宿主机 Linux 中安装针对目标机平台的交叉编译工具，并设置好交叉编译环境。
  - c) 设置宿主机 Linux 的 IP 地址
2. 搭建实验箱工作环境
  - a) 建立实验箱（目标机）与宿主机的物理连接（串口线、网线）。
  - b) 在宿主机上通过 minicom 远程登录、远程控制实验箱。
  - c) 设置实验箱中 Linux 的 IP 地址，并测试与宿主机之间的网络连通性。

- d) 测试是否能在宿主机上通过 `telnet` 远程登录、远程控制实验箱。
  - e) 检验是否能够通过 `tftp` 在宿主机和目标机之间传送文件。
3. 在宿主机上编写 C 语言程序，交叉编译后将可执行文件下载到目标机中测试执行效果。
  4. 搭建交叉调试环境，实验嵌入式程序的交叉调试过程。
  5. 在宿主机和目标机之间建立 SSH 远程登录环境和 `scp` 文件拷贝机制

## 四. 实验步骤

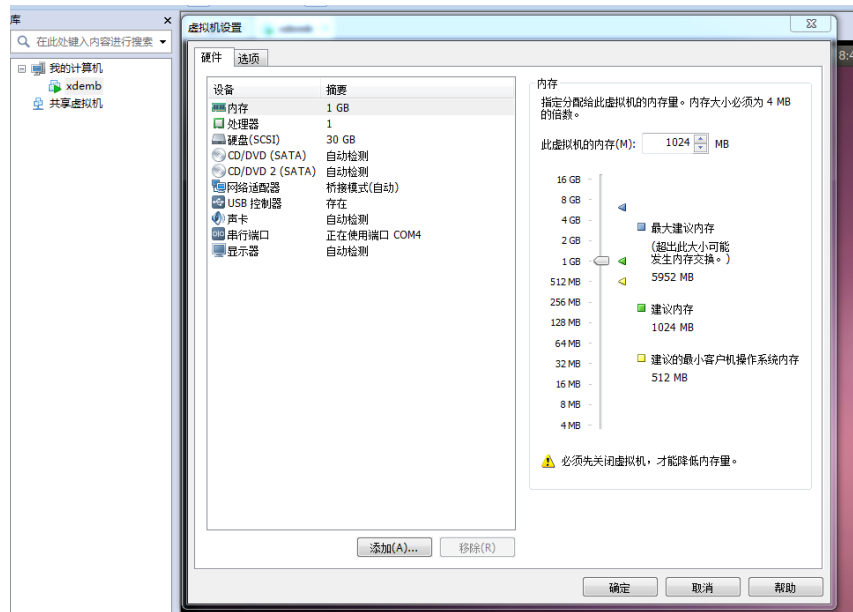
### 4.1 搭建宿主机工作环境

#### 1. 在宿主机（物理机或虚拟机）上安装 Linux 操作系统。

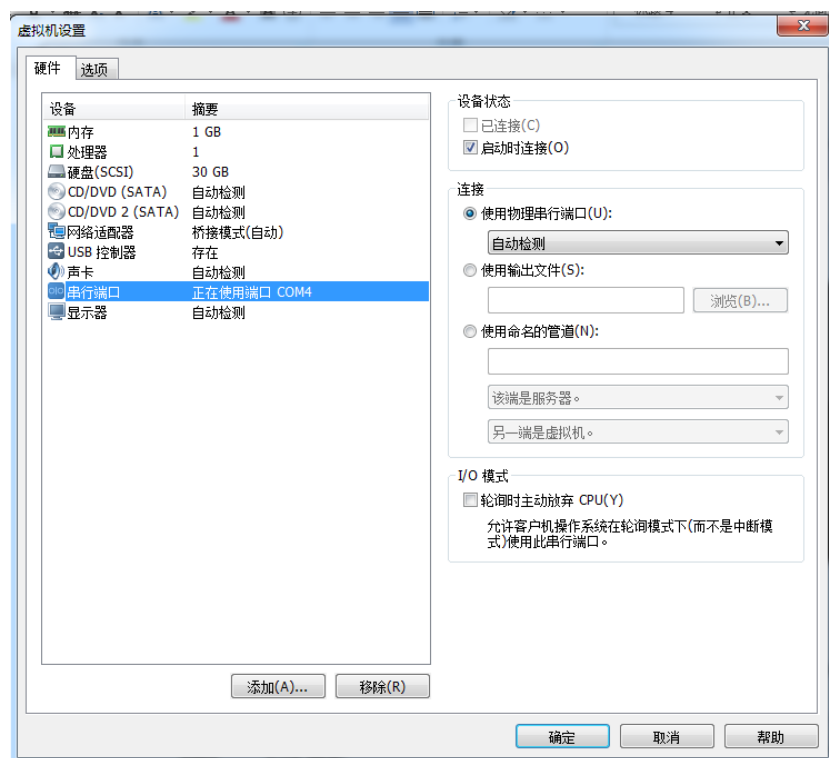
对于初学者而言，在虚拟机中安装 Linux 更加简单和安全，因此本次实验中我们选择在虚拟机中安装 Linux 操作系统。

**提示：** 为了缩短实验时间，我们已经在 VMWare 中安装好了一个 Ubuntu 系统，你只需要通过 Windows 桌面上的 VMWare 快捷方式直接启动“xdemb”虚拟机实例就可以了。但是在启动之前要检验该虚拟机设置是否正确。

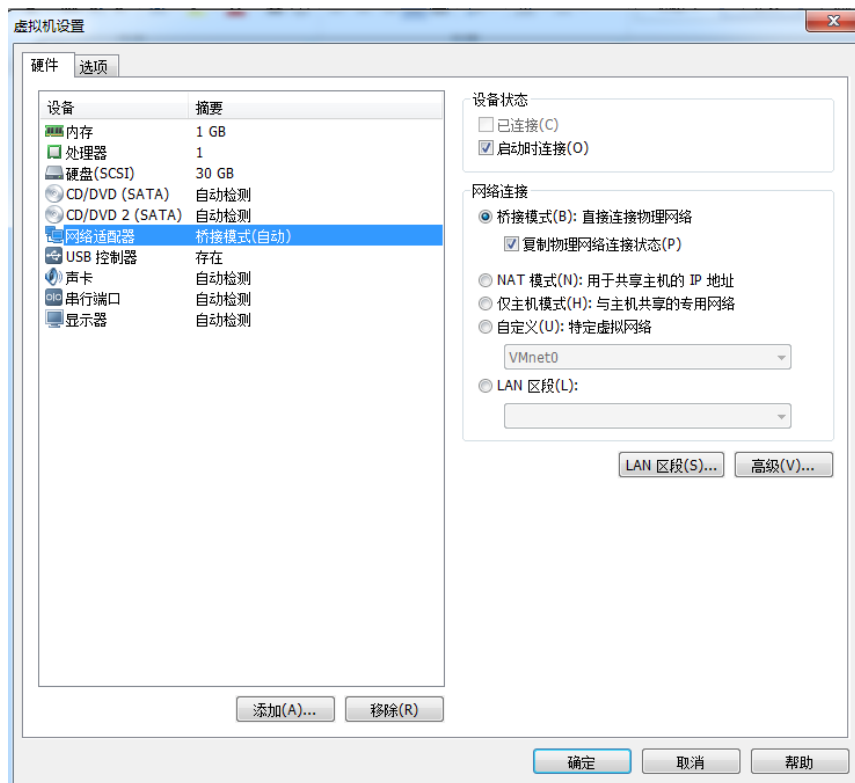
- a) 通过 Windows 桌面上的 VMWare 快捷方式启动 VMWare 软件。
- b) 右击“我的计算机>xdemb”，在弹出菜单中点击“设置”，出现如下对话框：



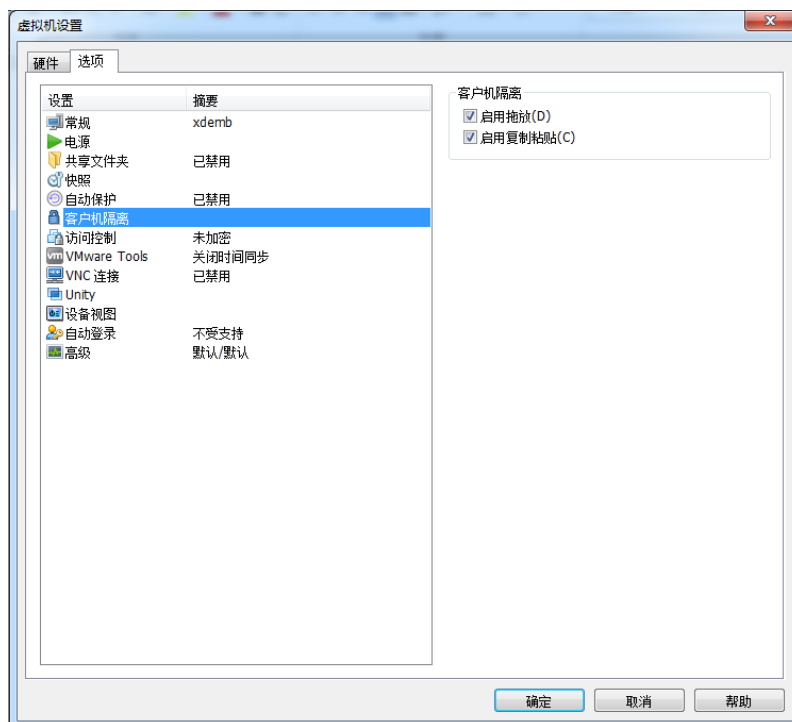
- c) 点击“串行端口”，弹出如下窗口，将虚拟机串口绑定到“COM1”或设为“自动检测”



- d) 点击“网络适配器”，弹出如下窗口，将虚拟机网卡设为“桥接模式”

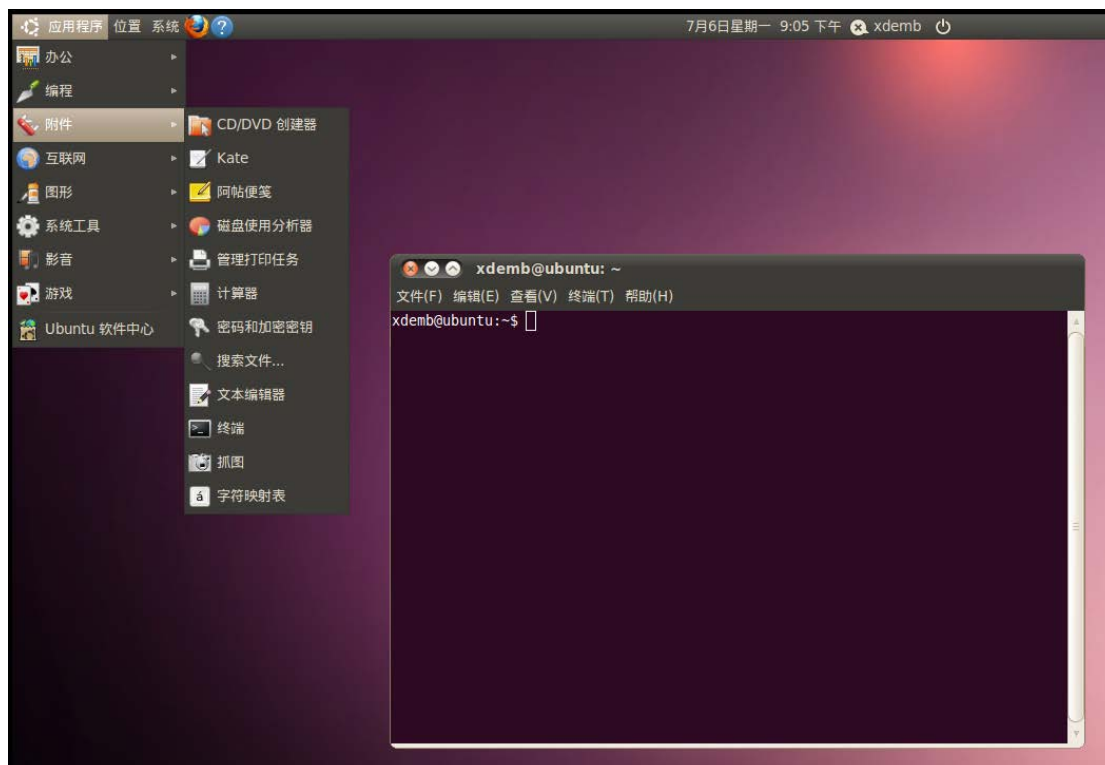


- e) 点击“选项”，点击“客户机隔离”，然后开启“启用拖放”和“启用复制粘贴”。这样做的目的是方便在 xdem b 虚拟机和 windows 之间传输文件。



- f) 启动“xdemb”虚拟机实例。
- g) 点击“xdemb”用户登录，登录密码为“123456”。

- h) 在 Linux 系统中，点击“应用程序>附件>终端”，将弹出 Linux 终端窗口，我们可以在终端中输入各种 Linux 命令，以完成指定任务。



2. 在宿主机 Linux 中安装针对目标机平台的交叉编译工具，并设置好交叉编译工具的查找路径。

针对特定的目标机平台生成交叉编译工具是一件很麻烦的事情！幸运的是我们已经将交叉编译工具生成完毕，并安装在了如下目录：

`/opt/cross-compiler/arm-2009q3/bin`

- a) 在“终端”中输入命令：

```
$cd /opt/cross-compiler/arm-2009q3/bin
```

将当前目录切换到`/opt/cross-compiler/arm-2009q3/bin`。

此时可以利用 `ls` 命令（“`ls`”、“`ls -l`”或“`ls -al`”）查看该目录下有哪些开发工具。

- b) 在“终端”中输入命令：

```
$/arm-none-linux-gnueabi-gcc -v
```

获得交叉编译器 `arm-none-linux-gnueabi-gcc` 的详细版本信息。



- c) 将/opt/cross-compiler/arm-2009q3/bin 添加到 PATH 环境变量中:

```
$export PATH=/opt/cross-compiler/arm-2009q3/bin:$PATH
```

- d) 利用“echo PATH”命令查看该环境变量是否设置成功。

- e) 在任意目录下输入命令

```
$arm-none-linux-gnueabi-gcc -v
```

或命令

```
$which arm-none-linux-gnueabi-gcc
```

检查改过之后的 PATH 环境变量是否生效。

- f) 为了避免每次开机都要重新设置 PATH 环境变量，可以将

“ export PATH=/opt/cross-compiler/arm-2009q3/bin:\$PATH ” 添加 到 /etc/bash.bashrc 文件的最后一行。具体方法为:

执行命令: `$sudo gedit /etc/bash.bashrc`

然后在 gedit 中将“export PATH=/opt/cross-compiler/arm-2009q3/bin:\$PATH” 添加到最后一行。

最后执行命令: `$source /etc/bash.bashrc`

### 3. 设置宿主机 Linux 系统的 IP 地址。

设置 Linux 系统 IP 地址的方法有两种，一种是通过 ifconfig 命令。

- a) 查看当前 IP:

```
$ifconfig
```

- b) 启动关闭指定网卡 eth2:

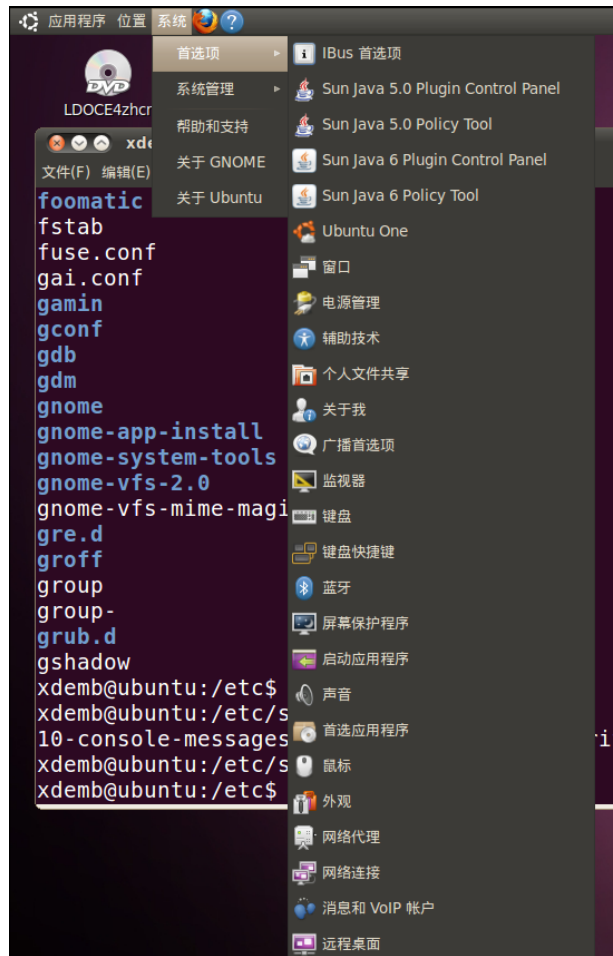
```
$ifconfig eth2 up
```

```
$ifconfig eth2 down
```

- c) 设置网卡 eth2 的 IP 地址和子网掩码:

```
$ifconfig eth2 192.168.0.101 netmask 255.255.255.0
```

设置 Linux 系统 IP 地址的另一种方法是利用图形用户界面，具体方法是点击 Ubuntu 桌面环境的“系统>首选项>网络连接”:



然后在如下窗口中选中要编辑的网卡（eth2）



点击“编辑>IPv4 设置”，然后将“方法”设为“手动”，最后点击“添加”按钮，添加 IP 地址。

在本次实验中我们需要将宿主机 Linux 系统的 IP 地址设为“192.168.0.101”。

## 4.2 搭建实验箱工作环境

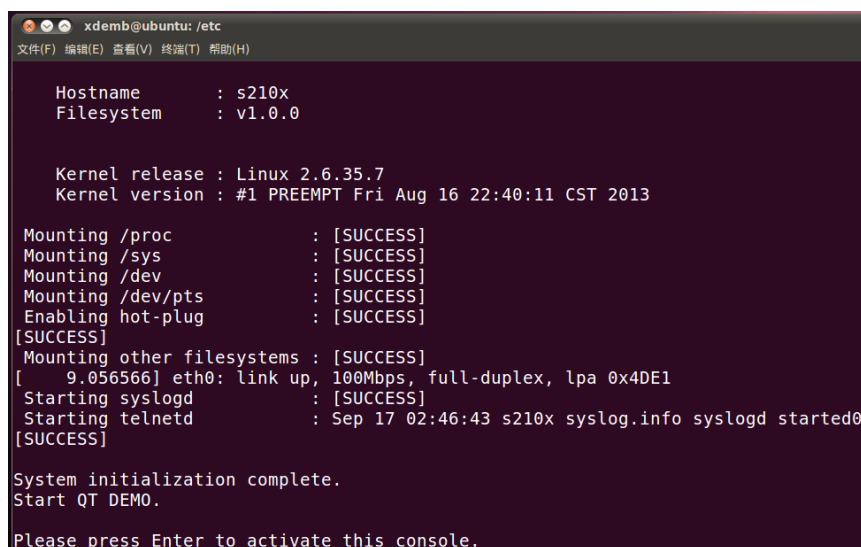
1. 建立实验箱（目标机）与宿主机的物理连接（串口线、网线）。
  - a) 关闭宿主机和实验箱的电源！（RS232 串口线不是热插拔的！）
  - b) 在宿主机和实验箱之间连接好串口线和网线。

2. 在宿主机上通过 **minicom** 远程登录、远程控制实验箱。

- a) 开启宿主机电源，启动 Ubuntu 虚拟机。
- b) 启动 Ubuntu 虚拟机中的“终端”工具。
- c) 在终端中输入命令：

```
$minicom
```

- d) 打开实验箱的电源，此时应该可以看到在 minicom 的终端窗口中有很多提示信息在快速显示和刷新！耐心等待直到 minicom 不再有新的信息出现并静止在如下画面：（此时实验箱的 Linux 已经正常启动）



```
xdemb@ubuntu: /etc
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

Hostname      : s210x
Filesystem    : v1.0.0

Kernel release : Linux 2.6.35.7
Kernel version : #1 PREEMPT Fri Aug 16 22:40:11 CST 2013

Mounting /proc      : [SUCCESS]
Mounting /sys       : [SUCCESS]
Mounting /dev       : [SUCCESS]
Mounting /dev/pts   : [SUCCESS]
Enabling hot-plug   : [SUCCESS]
[SUCCESS]
Mounting other filesystems : [SUCCESS]
[  9.056566] eth0: link up, 100Mbps, full-duplex, lpa 0x4DE1
Starting syslogd    : [SUCCESS]
Starting telnetd    : Sep 17 02:46:43 s210x syslog.info syslogd started
[SUCCESS]

System initialization complete.
Start QT DEMO.

Please press Enter to activate this console.
```

- e) 按下宿主机键盘上的“回车”键，进入如下画面：

```
xdemb@ubuntu: /etc
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

Kernel release : Linux 2.6.35.7
Kernel version : #1 PREEMPT Fri Aug 16 22:40:11 CST 2013

Mounting /proc      : [SUCCESS]
Mounting /sys       : [SUCCESS]
Mounting /dev       : [SUCCESS]
Mounting /dev/pts   : [SUCCESS]
Enabling hot-plug   : [SUCCESS]
[SUCCESS]
Mounting other filesystems : [SUCCESS]
[ 9.052680] eth0: link up, 100Mbps, full-duplex, lpa 0x4DE1
Starting syslogd     : [SUCCESS]
Starting telnetd     : Sep 17 02:56:06 s210x syslog.info syslogd started0
[SUCCESS]

System initialization complete.
Start QT DEMO.

Please press Enter to activate this console.
starting pid 79, tty '': '/bin/sh'
running /etc/profile
[root@s210x ~]#
```

其中 root 表示 root 用户，s210x 表示目标机 Linux 的主机名称，即以 root 身份登录到了 s210x 设备。

- f) 在 minicom 终端中输入 ls 命令就可以查看 s210x 实验箱中的文件系统。
- g) 也可以在 minicom 终端中输入各种命令远程控制实验箱。
- h) 如果在步骤 d)中看不到任何提示信息，则很有可能是 RS232 串口参数设置不正确。此时先从 minicom 中退出，然后输入命令：

`$minicom -s`

进入画面：

```
+-----[configuration]-----+
| Filenames and paths          |
| File transfer protocols      |
| Serial port setup           |
| Modem and dialing            |
| Screen and keyboard          |
| Save setup as dfl             |
| Save setup as..              |
| Exit                          |
| Exit from Minicom            |
+-----+-----+

```

- i) 然后将串口参数设为“115200 8N1”“无流控”，即采用如下设置。设置好之后再重复步骤 c)

```
+-----+-----+
| A - Serial Device      : /dev/ttyS0 |
| B - Lockfile Location  : /var/lock  |
| C - Callin Program     :            |
| D - Callout Program    :            |
| E - Bps/Par/Bits       : 115200 8N1 |
| F - Hardware Flow Control : No      |
| G - Software Flow Control : No      |
|                         |
| Change which setting?  |
+-----+-----+

```

### 3. 设置实验箱中 Linux 的 IP 地址，并测试与宿主机之间的网络连通性。

- a) 利用 `ifconfig` 命令将实验箱 Linux 系统 IP 地址设置为“192.168.0.101”。
- b) 利用 `ping` 命令测试实验箱 Linux 系统与宿主机 Linux 系统之间的网络连通性：

在 `minicom` 终端中输入命令：

```
$ping 192.168.0.100
```

或者在宿主机 Linux 系统中另外启动一个“终端”（本文之后将其称为“宿主机终端”），在宿主机终端中输入命令：

```
$ping 192.168.0.101
```

提示：提前终止 `ping` 命令以及其他 Linux 命令的方法是键入“CTRL+c”。

#### 4. 测试是否能在宿主机上通过 `telnet` 远程登录、远程控制实验箱。

- a) 在宿主机 Linux 系统中另外启动一个“终端”（本文之后将其称为“宿主机终端”）。
- b) 在宿主机终端中输入命令：

```
$telnet 192.168.0.101
```

然后输入用户名 `root`，密码是无或“123456”

- c) `telnet` 正确登录之后，也会出现“[root@s210x ~]#”的提示，表示以 `root` 身份登录 `s210x` 设备成功。
- d) 在 `telnet` 终端中输入 `ls` 命令就可以查看 `s210x` 实验箱中的文件系统。
- e) 也可以在 `telnet` 终端中输入各种命令远程控制实验箱。

#### 5. 检验是否能够通过 `tftp` 在宿主机和目标机之间传送文件。

虚拟机中 Ubuntu 系统已经默认安装了 `tftp` 服务器端软件，并且将 `tftp` 工作目录设在了“`/opt/tftp`”。如果要将某文件 `file1` 下载到实验箱中，可以先将 `file1` 拷贝到 `/opt/tftp` 目录，然后在实验箱终端（即 `minicom` 终端或 `telnet` 终端）中利用 `tftp` 命令下载到实验箱的指定目录。

`tftp` 命令用法为：

- a) 下载文件

```
tftp 服务器地址 IP 地址 -g -r 服务器端源文件名 -l 本地文件名
```

测试是否能够通过 `tftp` 在宿主机和目标机之间传送文件的具体方法如下：

在宿主机终端中输入命令：

- a) `$cd /opt/tftp`
- b) `$touch file1`
- c) `$gedit file1`
- d) 然后在 gedit 编辑器中随便为 file1 录入一些文字并保存退出。

在目标机终端（minicom 终端或 telnet 终端）中输入命令：

- a) `#tftp 192.168.0.100 -g -r ./file1 -l ./file1`
- b) 用 `vi file1` 命令查看下载的文件内容是否与原始文件一致。

### 4.3 交叉编译并下载运行可执行程序

在宿主机上编写 C 语言程序，交叉编译后将可执行文件下载到目标机中测试执行效果。具体实验过程为：

1. 在宿主机中建立/home/xdemb/projects/helloworld 目录：  
`$mkdir -p /home/xdemb/projects/helloworld`
2. 在/home/xdemb/projects/hello 目录中利用 vi 或 gedit 工具创建附录 1 的 C 语言程序 hello.c。
3. 编译 hello.c。
  - a) 编译方法一是用命令：  
`$arm-none-linux-gnueabi-gcc -o hello hello.c`
  - b) 编译方法二是在/home/xdemb/projects/helloworld 目录下建立附录 2 的 Makefile 文件，并输入命令：  
`$make`  
还可以利用 “make install”命令将生成结果拷贝到/opt/tftp 目录；或者利用 “make clean”清除编译结果。
4. 编译成功之后可以在/home/xdemb/projects/helloworld 目录下发现一个新的文件 hello，可以通过命令：  
`$file hello`  
观察 hello 文件的文件属性（针对 ARM 平台的可执行程序）

5. 将 `hello` 可执行程序拷贝到 `/opt/tftp`。
6. 通过 `minicom` 或 `telnet` 登录到实验箱（目标机）。
7. 利用 `tftp` 命令将宿主机的 `hello` 可执行程序下载到实验箱 Linux 系统的 `/home/app` 目录。

```
#cd /home/app
#tftp 192.168.0.100 -g -r ./hello -l ./hello
#chmod 777 hello
```

上面使用 `chmod` 命令的目的是将 `hello` 设置为可执行程序。（`tftp` 下载的任何程序默认没有执行权限。）

8. 通过 `minicom` 或 `telnet` 终端在目标机中执行 `hello` 程序：

```
#!/hello
```

9. 执行成功将在 `minicom` 终端中看到如下结果：

```
[root@s210x /home/app]#./hello
HelloWorld!
HelloWorld!
HelloWorld!
HelloWorld!
HelloWorld!
HelloWorld!
[root@s210x /home/app]#
```

## 4.4 交叉调试实验

### 1. 建立交叉调试环境。

进行交叉调试时，需要在目标机上运行 `gdbserver` 调试软件和被调试程序，在宿主机上运行 `gdb` 调试软件（客户端方式运行）。在宿主机上通过 `gdb` 客户端软件向 `gdbserver` 发送调试命令。

在本实验中，宿主机上的 `gdb` 调试软件已经正常安装（`/opt/cross-compiler/arm-2009q3/bin` 目录下的 `arm-none-linux-gnueabi-gcc` 程序）。但是，目标机上并没有 `gdbserver` 软件！因此我们必须首先在目标机中安装 `gdbserver` 软件。

具体步骤如下：

- a) 我们已经在 Ubuntu 虚拟机的 `/opt/tools` 目录下存放了 `gdb` 软件的源代码：`gdb-6.8a.tar.gz`。首先将其解压：

```
$cd /opt/tools
```

```
$tar zxvf gdb-6.8.tar.gz
```

- b) 配置 gdbserver 源码的交叉编译参数。

```
$cd gdb-6.8/gdb/gdbserver
```

```
$./configure CC=arm-none-linux-gnueabi-gcc --host=i686-pc-linux-gnu  
--target=arm-none-linux-gnueabi
```

- c) 交叉编译生成 gdbserver

```
$make
```

- d) 如果编译成功，则在 gdb-6.8/gdb/gdbserver 目录下可以看到新生成的 gdbserver 可执行程序。

- e) 将 gdbserver 拷贝到/opt/tftp 目录。

- f) 通过 minicom 或 telnet 登录到目标机，并利用 tftp 命令将 gdbserver 下载到实验箱的/bin 目录。

```
#tftp 192.168.0.100 -r ./gdbserver -l /bin/gdbserver
```

## 2. 交叉调试 hello 程序

- a) 重新编译附录 1 的 hello.c 程序（增加了 -g 编译参数）

```
$arm-none-linux-gnueabi-gcc -g -o hello hello.c
```

- b) 将新生成的 Debug 版的 hello 程序利用 tftp 下载到目标机。

- c) 在目标机上启动 gdbserver:

```
#gdbserver 192.168.0.101:1234 ./hello
```

- d) 在宿主机上启动 arm-none-linux-gnueabi-gdb:

```
$arm-none-linux-gnueabi-gdb ./hello
```

此时进入 gdb 的命令行模式。

- e) 在 gdb 中输入如下命令以设置 so 文件和 lib 文件的查找路径:

```
(gdb)set solib-search-path /opt/cross-compiler/arm-2009q3/arm-none-linux-  
gnueabi/libc/lib/
```

- f) 在 gdb 中通过 target remote 命令连接到目标机的 1234 端口:

```
(gdb)target remote 192.168.0.101:1234
```

- g) 连接成功后，利用 gdb 的 “list”命令显示 hello.c 程序源码，然后利用



“break 8”命令在 `hello.c` 的第 8 行（`printf("HelloWorld!\n");` 语句处）加入断点，最后利用“continue”命令远程执行 `hello` 程序，并观察程序是否在断点处中断。可以利用“print i”命令查看变量 `i` 的值。

- h) 利用 “quit”命令从 `arm-none-linux-gnueabi-gdb` 退出。服务器端也自动退出。

## 4.5 搭建 SSH 远程登录环境

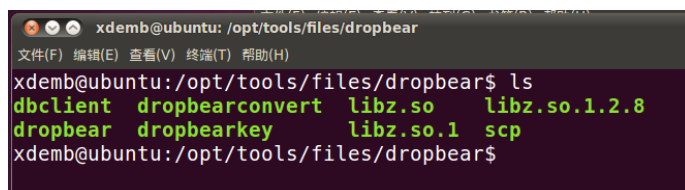
SSH 是 Secure Shell 的缩写，是专门为以安全方式远程登录网络主机设计的一套协议。SSH 系统由客户端和服务端软件组成。在这里，我们将实验箱（目标机）作为 SSH 的服务器端，宿主机（Ubuntu 虚拟机）作为 SSH 的客户端。

在虚拟机中安装 Ubuntu 之后，已经默认安装了 SSH 的客户端软件，只需要在终端中输入 `ssh` 命令就可以启动 SSH 的客户端软件。

但是，在目标机中并没有部署 SSH 的服务器端软件。因此我们需要再目标机中移植一个 SSH 服务器端软件。我们选择了嵌入式 Linux 环境中常用的 SSH 服务器软件 `dropbear`（下载地址：<https://matt.ucc.asn.au/dropbear/releases/dropbear-2015.67.tar.bz2>）。

在目标机中移植的 `dropbear` 的通常做法是将其源码包解压，然后交叉编译（`dropbear` 依赖于一个第三方库 `zlib`，因此需要先交叉编译 `zlib` 库，其下载地址为 <http://www.zlib.net/>），然后将编译结果利用 `tftp` 下载到目标机。这是一个繁琐而且容易出错的过程，因此这里直接将我们的编译结果提供给实验者。（有兴趣的同学可以试着交叉编译并部署 `dropbear`，虽然麻烦，但可以学习到很多东西。）

交叉编译生成的 `dropbear` 软件及其依赖库被存放在 `/opt/tools/files/dropbear` 目录中，包含如下文件：



```
xdemb@ubuntu: /opt/tools/files/dropbear
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
xdemb@ubuntu:/opt/tools/files/dropbear$ ls
dbclient  dropbearconvert  libz.so  libz.so.1.2.8
dropbear  dropbearkey      libz.so.1  scp
xdemb@ubuntu:/opt/tools/files/dropbear$
```

在实验箱中部署 `dropbear` 的过程为：

1. 用 tftp 将 dropbear 下载到目标机的/sbin 目录（注意用 chmod 更改可执行属性）；
2. 用 tftp 将 dropbearkey 下载到目标机的/bin 目录（注意用 chmod 更改可执行属性）；
3. 用 tftp 将 dropbearconvert 下载到目标机的/bin 目录（注意用 chmod 更改可执行属性）；
4. 用 tftp 将 dbclient 下载到目标机的/usr/bin 目录（注意用 chmod 更改可执行属性）（/usr/bin 目录可能不存在，需要用 mkdir 创建）；
5. 用 tftp 将 scp 下载到目标机的/bin 目录（注意用 chmod 更改可执行属性）；
6. 将三个库文件 libz.so、libz.so.1、libz.so.1.2.8 下载到目标机的/lib 目录；
7. 在目标机中建立/etc/dropbear 目录：

```
#mkdir -p /etc/dropbear
```

8. 在/etc/dropbear 目录中建立 ssh 所需的公、私钥文件：

```
#dropbearkey -t rsa -f dropbear_rsa_host_key
```

```
#dropbearkey -t dss -f dropbear_dss_host_key
```

```
#dropbearkey -t ecdsa -f dropbear_ecdsa_host_key
```

9. 在目标机中利用 vi 工具在/etc/init.d/rcS 文件最后一行加入“dropbear”，这样每次目标机加电重启之后，ssh 服务端软件 dropbear 都会自动执行。
10. 需要按照如下方法将目标机 root 用户的口令设为“123456”：

a) 

```
#cat /etc/passwd > /etc/shadow
```

b) 

```
#passwd root
```

 （根据提示将口令设为“123456”）

c) 利用 vi 工具打开/etc/passwd 文件，并将其内容删除干净

d) 

```
#cat /etc/shadow > /etc/passwd
```

经过以上复杂过程，在目标机中已经成功部署了 ssh 服务器端环境。现在实验在宿主机中是否能够通过 ssh 登录目标机：

1. 首次 ssh 登录需要用 rm 命令将/home/xdemb/.ssh 子目录下的“known\_hosts”文件删除。
2. 利用 ssh 远程登录目标机：

```
$ssh root@192.168.0.101
```

忽略相关警告信息，并输入密码 123456 就可以登录到目标机。

SSH 环境部署成功之后也可以通过 scp 命令在宿主机和目标机之间相互拷贝文件。scp 的用法如下：

```
$scp local_file remote_username@remote_ip:remote_folder
```

## 五. 附录：

### 附录 1：hello.c 程序

```
/* file: hello.c */
#include <stdio.h>
int main(int argc, char **argv)
{
    int i;
    for (i=0; i<5; i++)
    {
        printf("HelloWorld!\n");
    }
    return 0;
}
```

### 附录 2：Makefile 文件

```
INSTALLDIR = /opt/tftp

#----- /* execute file(s) */
TESTFILE   = hello
#----- /* object file(s) */
SRCFILE    = hello.c
#----- /* header file(s) */
TESTFILE_H =

CROSS = arm-none-linux-gnueabi-
CC = $(CROSS)gcc
```

```
AS = $(CROSS)as
```

```
LD = $(CROSS)ld
```

```
CFLAGS += -O2 -Wall
```

```
all: $(TESTFILE)
```

```
$(TESTFILE): $(SRCFILE) $(TESTFILE_H) Makefile
```

```
$(CC) $(CFLAGS) -o $@ $@.c -static
```

```
clean:
```

```
rm -f $(TESTFILE)
```

```
install: $(TESTFILE)
```

```
mkdir -p $(INSTALLDIR)
```

```
cp --target-dir=$(INSTALLDIR) $(TESTFILE)
```

# 实验二 嵌入式 Web 服务器软件的实现

## 一. 实验目的

1. 熟悉 Linux 网络编程。
2. 了解 HTTP 协议和 Web 服务器工作原理。
3. 掌握嵌入式 Linux 中用多进程、多线程、I/O 多路复用三种方式实现并发网络服务器的方法。

## 二. 实验内容

1. 用多进程方式实现 Web 服务器。
2. 用多线程方式实现 Web 服务器。
3. 用 I/O 多路复用方式实现 Web 服务器。

## 三. 实验步骤:

### 3.1 环境配置

1. 打开实验箱，用网线和串口线连接宿主机，并连接实验箱电源线。
2. 启动虚拟机中的 Linux 系统，在终端中运行 minicom 命令，启动 minicom 软件

```
Welcome to minicom 2.4

OPTIONS: I18n
Compiled on Jan 25 2010, 06:49:09.
Port /dev/ttyS0

Press CTRL-A Z for help on special keys

□
```

3. 拨动实验开发板的电源开关，启动目标机，进入嵌入式 Linux 系统。嵌入式 Linux 系统启动完毕之后按下宿主机的回车键。

```
System initialization complete.
Oct 13 16:07:44 s210x authpriv.info dropbear[83]: Running in background

Please press Enter to activate this console.
starting pid 84, tty '': '/bin/sh'
running /etc/profile
[root@s210x ~]#
```

4. 用 `ifconfig` 查看目标机的 ip 地址，设为 192.168.0.101,然后用 `ping` 命令来测试目标机和宿主机到连通性。

```
[root@s210x ~]#ifconfig eth0 192.168.0.101 netmask 255.255.255.0
[root@s210x ~]#ping 192.168.0.7 -c 2
PING 192.168.0.7 (192.168.0.7): 56 data bytes
64 bytes from 192.168.0.7: seq=0 ttl=64 time=6.013 ms
64 bytes from 192.168.0.7: seq=1 ttl=64 time=0.403 ms

--- 192.168.0.7 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.403/3.208/6.013 ms
[root@s210x ~]#
```

5. 把程序源代码文件拷贝到宿主机的 `/opt/nfs/web` 目录下或者直接在此目录下创建并编辑源代码文件，结果如下。

```
root@ubuntu:/# ls /opt/nfs/web/
web_server_process.c  web_server_select.c  web_server_thread.c
root@ubuntu:/#
```

## 3.2 实现多进程 Web 服务器

1. 用 `arm-none-linux-gnueabi-gcc` 命令编译源程序，得到可执行程序 `web_server_process`。

```
root@ubuntu:/# cd /opt/nfs/web/
root@ubuntu:/opt/nfs/web# arm-none-linux-gnueabi-gcc web_server_process.c -o web_server_process
root@ubuntu:/opt/nfs/web# ls
web_server_process  web_server_process.c  web_server_select.c  web_server_thread.c
root@ubuntu:/opt/nfs/web#
```

2. 用 `vi` 文本编译器创建文件 `index.html`，用于测试 Web 服务器。

```
$vi index.html
```

3. 在 `index.html` 文件中加入如下内容：

```
<html>
  <head><title>Hello World</title></head>
  <body> Welcome to our new Web Server </body>
</html>
~
~
~
~
~
```

4. 在 `/opt/nfs/web` 目录创建 `cgi-bin` 目录，并在新创建的目录下用 `vi` 编辑器创建文件 `hello.cgi`

```
root@ubuntu:/opt/nfs/web# mkdir cgi-bin
root@ubuntu:/opt/nfs/web# vi cgi-bin/hello.cgi
```

5. hello.cgi 的内容如下图。

```
#!/bin/sh
echo "Content-type:text/plain"
echo
echo " Welcome to Web Server"
echo " This is a cgi file."

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

:wa

6. 用 `ls -l` 命令可以看到其他用户没有执行权限，通过 `chmod` 命令来修改 `hello.cgi` 的权限，使它成为可执行文件。

```
root@ubuntu:/opt/nfs/web/cgi-bin# chmod o+x hello.cgi
root@ubuntu:/opt/nfs/web/cgi-bin# ls -l
总用量 4
-rw-r--r-x 1 root root 104 2015-05-05 23:21 hello.cgi
root@ubuntu:/opt/nfs/web/cgi-bin#
```

7. 使用 `service` 命令来重启宿主机上的 `nfs` 服务器 (`nfs-kernel-server`), 用 `exportfs` 命令查看它的工作目录。如图所示

```
root@ubuntu:/opt/nfs/web# service nfs-kernel-server restart
* Stopping NFS kernel daemon
* Unexporting directories for NFS kernel daemon...
* Exporting directories for NFS kernel daemon...
* Starting NFS kernel daemon
root@ubuntu:/opt/nfs/web# exportfs -v
/opt/nfs          <world>(rw,wdelay,root_squash,no_subtree_check)
root@ubuntu:/opt/nfs/web#
```

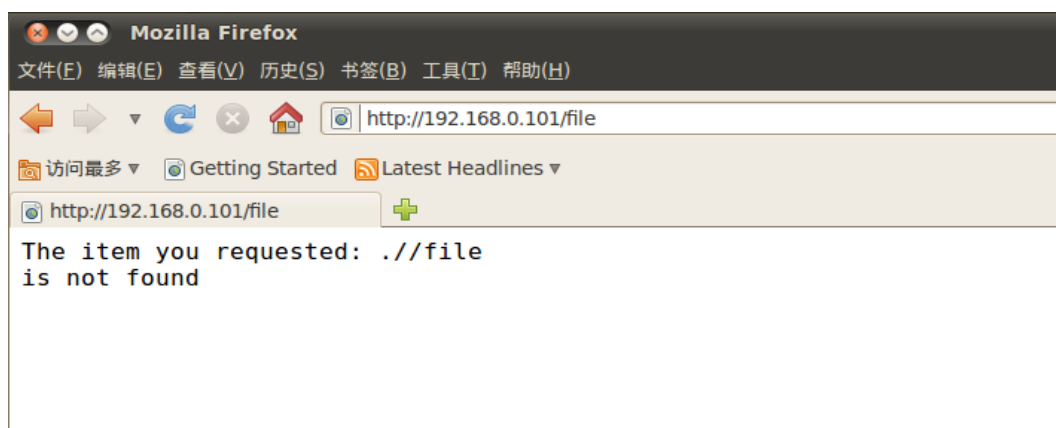
8. 在目标机上挂载 nfs,如下图。

```
[root@s210x ~]#mount -t nfs -o nolock,rsize=1024,wsiz=1024 192.168.0.7:/opt/nfs
/mnt/usb
[root@s210x ~]#ls /mnt/usb/web/
cgi-bin                web_server_process     web_server_select.c
index.html              web_server_process.c   web_server_thread.c
[root@s210x ~]#
```

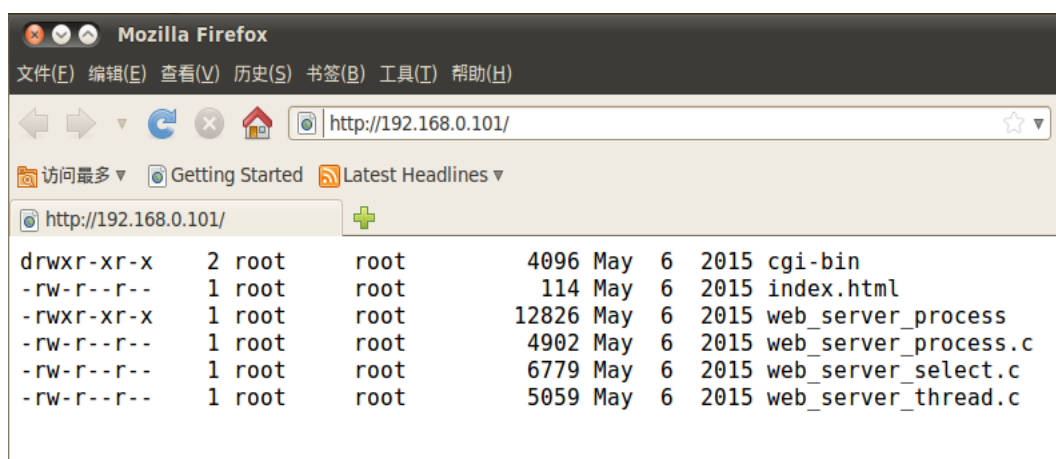
9. 在目标机上运行 web\_server\_process.

```
[root@s210x ~]#cd /mnt/usb/web/  
[root@s210x /mnt/usb/web]#./web_server_process 80
```

10. 在宿主机中打开浏览器，输入 <http://192.168.0.101/file> 查看结果。



11. 在宿主主机的浏览器输入，<http://192.168.0.101> 查看结果



12. 在宿主主机的浏览器输入，<http://192.168.0.101/index.html> 查看结果

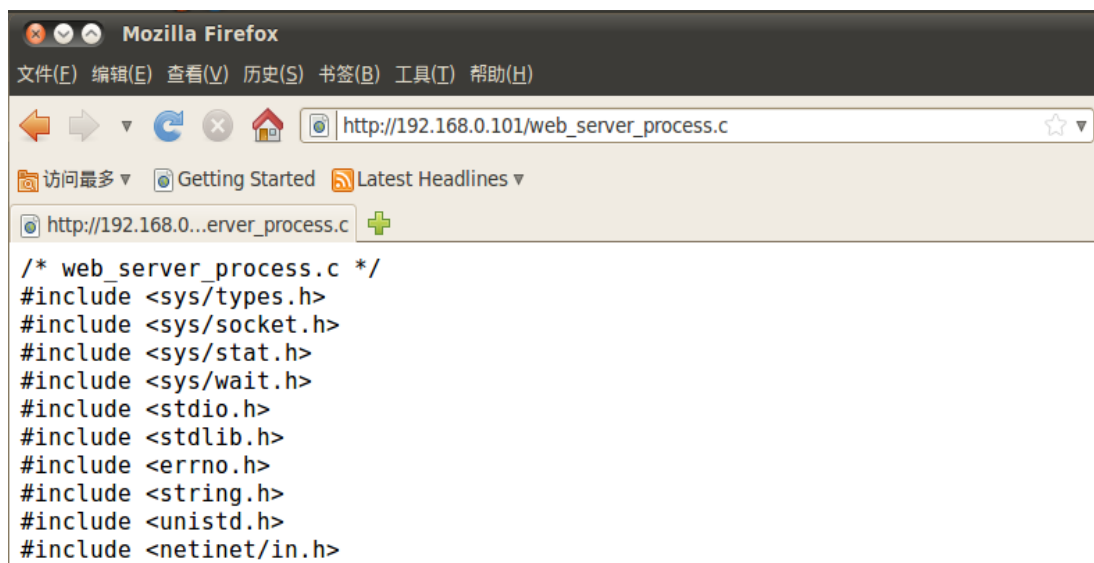


13. 在宿主主机的浏览器输入，<http://192.168.0.101/cgi-bin/hello.cgi> 查看结果





14)在宿主主机的浏览器输入, [http://192.168.0.101/web\\_server\\_process.c](http://192.168.0.101/web_server_process.c) 查看结果



### 3.3 实现多线程 Web 服务器

1. 在宿主机终端用 arm-none-linux-gnueabi-gcc 编译源程序, 得到可执行程序 web\_server\_thread

```
root@ubuntu:/opt/nfs/web# arm-none-linux-gnueabi-gcc web_server_thread.c -o web_server_thread -lpthread
root@ubuntu:/opt/nfs/web# ls
cgi-bin      web_server_process  web_server_select.c  web_server_thread.c
index.html   web_server_process.c  web_server_thread
```

2. 在目标机上运行 web\_server\_thread

```
[root@s210x /mnt/usb/web]# ./web_server_thread 80
```

3. 在宿主主机上打开浏览器, 输入 <http://192.168.0.101> 查看结果。

4. 在宿主机上的浏览器中输入 <http://192.168.0.101/cgi-bin/hello.cgi>，查看结果。
5. 在宿主机上的浏览器中输入 [http://192.168.0.101/web\\_server\\_thread.c](http://192.168.0.101/web_server_thread.c)，查看结果。

### 3.3 实现 I/O 多路复用方式的 Web 服务器

1. 在宿主机上用 `arm-none-linux-gnueabi-gcc` 命令来编译 `web_server_select.c`

```
root@ubuntu:/opt/nfs/web# arm-none-linux-gnueabi-gcc web_server_select.c -o web_server_select
root@ubuntu:/opt/nfs/web# ls
cgi-bin      web_server_process  web_server_select  web_server_thread
index.html   web_server_process.c web_server_select.c web_server_thread.c
root@ubuntu:/opt/nfs/web#
```

2. 在目标机上运行刚编译的程序 `web_server_select`，端口一样为 80

```
[root@s210x /mnt/usb/web]# ./web_server_select 80
```

3. 在宿主机上打开浏览器输入 <http://192.168.0.101> 查看结果
4. 在宿主机上的浏览器中输入 <http://192.168.0.101/cgi-bin/hello.cgi>，查看结果。
5. 在宿主机上的浏览器中输入 [http://192.168.0.101/web\\_server\\_select.c](http://192.168.0.101/web_server_select.c)

## 四. 附录

### 附录 1. 多进程 Web 服务器

```
/* web_server_process.c */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
```

```

#include <unistd.h>
#include <netinet/in.h>

#define BUFSIZE 1024
#define MAX_QUE_CONN_NM 10

int start_server(int portnum);
void sigchld_handler(int sig);
int process_request(char *rq, int fd);
int resp_cannot_do(int fd);
int not_exist(char *f);
int resp_do_404(char *item, int fd);
int is_a_dir(char *f);
int resp_do_ls(char *dir, int fd);
int is_a_cgi_file(char *f);
int resp_do_exec(char *prog, int fd);
int resp_do_cat(char *f, int fd);
void pack_header(FILE *fp, char *extension);
char *get_filename_extension(char *f);
void ignore_others(FILE *fp);

int main(int argc, char *argv[])
{
    int sock;
    int portnum;
    int fd;
    pid_t pid;
    FILE *fpin;
    char request[BUFSIZE];

    if (argc != 2) {
        fprintf(stderr, "usage: %s <portnum>\n", argv[0]);
        exit(-1);
    }
    portnum = atoi(argv[1]);
    sock = start_server(portnum);
    signal(SIGCHLD, sigchld_handler);

    while(1) {
        fd = accept(sock, NULL, NULL);

        while ((pid = fork()) == -1);
        if (pid == 0) {
            fpin = fdopen(fd, "r");

```

```

        memset(request, 0, sizeof(request));
        fgets(request, BUFSIZE, fpin);
        ignore_others(fpin);
        process_request(request, fd);
        fclose(fpin);
        exit(0);
    }
    close(fd);
}
close(sock);
exit(0);
}

```

```
int start_server(int portnum)
```

```

{
    struct sockaddr_in server_sockaddr;
    int sockfd;
    int i = 1;

    /* create socket link */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(-1);
    }

    server_sockaddr.sin_family = AF_INET;
    server_sockaddr.sin_port = htons(portnum);
    server_sockaddr.sin_addr.s_addr = INADDR_ANY;
    bzero(server_sockaddr.sin_zero, 8);

    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &i, sizeof(i));

    if (bind(sockfd, (struct sockaddr *)&server_sockaddr, sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(-1);
    }

    if (listen(sockfd, MAX_QUE_CONN_NM) == -1)
    {
        perror("listen");
        exit(-1);
    }

    return sockfd;
}

```

```

}

void sigchld_handler(int sig)
{
    while (waitpid(-1, 0, WNOHANG) > 0);
    return;
}

int process_request(char *rq, int fd)
{
    char cmd[BUFSIZE];
    char arg[BUFSIZE];

    strcpy(arg, "/");
    if (sscanf(rq, "%s %s", cmd, arg + 2) != 2) {
        return 1;
    }

    if (strcmp(cmd, "GET") != 0)
    {
        resp_cannot_do(fd);
    } else if (not_exist(arg)) {
        resp_do_404(arg, fd);
    } else if (is_a_dir(arg)) {
        resp_do_ls(arg, fd);
    } else if (is_a_cgi_file(arg)) {
        resp_do_exec(arg, fd);
    } else {
        resp_do_cat(arg, fd);
    }

    return 0;
}

int resp_cannot_do(int fd)
{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "That command is not yet implemented\r\n");
    fclose(fp);
    return 0;
}

```

```

int not_exist(char *f)
{
    struct stat info;
    return (stat(f, &info) == -1);
}

int resp_do_404(char *item, int fd)
{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 404 Not Found\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);
    fclose(fp);
    return 0;
}

int is_a_dir(char *f)
{
    struct stat info;
    return (stat(f, &info) != -1 && S_ISDIR(info.st_mode));
}

int resp_do_ls(char *dir, int fd)
{
    FILE *fp = fdopen(fd, "w");
    pack_header(fp, "text/plain");
    fprintf(fp, "\r\n");
    fflush(fp);

    dup2(fd, 1);
    dup2(fd, 2);
    close(fd);
    execlp("ls", "ls", "-l", dir, NULL);
    perror(dir);
    return 1;
}

int is_a_cgi_file(char *f)
{
    return(strcmp(get_filename_extension(f), "cgi") == 0);
}

```

```

int resp_do_exec(char *prog, int fd)
{
    FILE *fp = fdopen(fd, "w");
    pack_header(fp, NULL);
    fflush(fp);

    dup2(fd, 1);
    dup2(fd, 2);
    close(fd);
    execl(prog, prog, NULL);
    perror(prog);
    return 1;
}

int resp_do_cat(char *f, int fd)
{
    int c;
    char *extension = get_filename_extension(f);
    char *content = "text/plain";
    FILE *fpsock, *fpfile;

    if (strcmp(extension, "html") == 0) {
        content = "text/html";
    } else if (strcmp(extension, "gif") == 0) {
        content = "text/gif";
    } else if (strcmp(extension, "jpg") == 0) {
        content = "text/jpg";
    } else if (strcmp(extension, "jpeg") == 0) {
        content = "text/jpeg";
    }

    fpsock = fdopen(fd, "w");
    fpfile = fopen(f, "r");
    if (fpsock != NULL && fpfile != NULL) {
        pack_header(fpsock, content);
        fprintf(fpsock, "\r\n");

        while ((c = getc(fpfile)) != EOF) {
            putc(c, fpsock);
        }
        fclose(fpfile);
        fclose(fpsock);
        return 0;
    }
}

```

```

        return 1;
    }

void pack_header(FILE *fp, char *extension)
{
    fprintf(fp, "HTTP/1.1 200 OK\r\n");
    if (extension)
        fprintf(fp, "Content-type: %s\r\n", extension);
}

char *get_filename_extension(char *f)
{
    return strchr(f, '.') + 1;
}

void ignore_others(FILE *fp)
{
    char buf[BUFSIZE];

    fgets(buf, BUFSIZE, fp);
    while(strcmp(buf, "\r\n"))
        fgets(buf, BUFSIZE, fp);
    return;
}

```

## 附录 2. 多线程 Web 服务器

```

/* web_server_thread.c */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <pthread.h>

#define BUFSIZE 1024
#define MAX_QUE_CONN_NM 10

```



```

void *thread(void *vargp);
int start_server(int portnum);
int process_request(int fd);

int resp_cannot_do(int fd);
int not_exist(char *f);
int resp_do_404(char *item, int fd);
int is_a_dir(char *f);
int resp_do_ls(char *dir, int fd);
int is_a_cgi_file(char *f);
int resp_do_exec(char *prog, int fd);
int resp_do_cat(char *f, int fd);
void pack_header(FILE *fp, char *extension);
char *get_filename_extension(char *f);
void ignore_others(FILE *fp);

int main(int argc, char *argv[])
{
    int sock;
    int portnum;
    int *fd;
    pthread_t tid;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <portnum>\n", argv[0]);
        exit(-1);
    }
    portnum = atoi(argv[1]);
    sock = start_server(portnum);

    while(1) {
        fd = malloc(sizeof(int));
        *fd = accept(sock, NULL, NULL);
        pthread_create(&tid, NULL, thread, fd);
    }
    close(sock);
    exit(0);
}

int start_server(int portnum)
{
    struct sockaddr_in server_sockaddr;
    int sockfd;
    int i = 1;

```

```

/* create socket link */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(-1);
}

server_sockaddr.sin_family = AF_INET;
server_sockaddr.sin_port = htons(portnum);
server_sockaddr.sin_addr.s_addr = INADDR_ANY;
bzero(server_sockaddr.sin_zero, 8);

setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &i, sizeof(i));

if (bind(sockfd, (struct sockaddr *)&server_sockaddr, sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(-1);
}

if (listen(sockfd, MAX_QUE_CONN_NM) == -1)
{
    perror("listen");
    exit(-1);
}

return sockfd;
}

void *thread(void *vargp)
{
    int fd = *((int *)vargp);
    pthread_detach(pthread_self());
    free(vargp);

    process_request(fd);
    close(fd);
    return;
}

int process_request(int fd)
{
    char request[BUFSIZE], cmd[BUFSIZE], arg[BUFSIZE];
    FILE *fpin;

```

```

    fpin = fdopen(fd, "r");
    fgets(request, BUFSIZE, fpin);
    ignore_others(fpin);

    strcpy(arg, "/");
    if (sscanf(request, "%s %s", cmd, arg + 2) != 2) {
        return 1;
    }

    if (strcmp(cmd, "GET") != 0)
    {
        resp_cannot_do(fd);
    } else if (not_exist(arg)) {
        resp_do_404(arg, fd);
    } else if (is_a_dir(arg)) {
        resp_do_ls(arg, fd);
    } else if (is_a_cgi_file(arg)) {
        resp_do_exec(arg, fd);
    } else {
        resp_do_cat(arg, fd);
    }

    fclose(fpin);
    return 0;
}

int resp_cannot_do(int fd)
{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "That command is not yet implemented\r\n");
    fclose(fp);
    return 0;
}

int not_exist(char *f)
{
    struct stat info;
    return (stat(f, &info) == -1);
}

int resp_do_404(char *item, int fd)

```

```

{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 404 Not Found\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);
    fclose(fp);
    return 0;
}

```

```

int is_a_dir(char *f)
{
    struct stat info;
    return (stat(f, &info) != -1 && S_ISDIR(info.st_mode));
}

```

```

int resp_do_ls(char *dir, int fd)
{
    pid_t pid;
    FILE *fp = fdopen(fd, "w");

    pack_header(fp, "text/plain");
    fprintf(fp, "\r\n");
    fflush(fp);

    while ((pid = fork()) == -1);
    if (pid == 0) {
        dup2(fd, 1);
        dup2(fd, 2);
        close(fd);

        execlp("ls", "ls", "-l", dir, NULL);
        perror(dir);
        exit(-1);
    }

    waitpid(pid, NULL, 0);
    return 0;
}

```

```

int is_a_cgi_file(char *f)
{
    return(strcmp(get_filename_extension(f), "cgi") == 0);
}

```

```

int resp_do_exec(char *prog, int fd)
{
    pid_t pid;
    FILE *fp = fdopen(fd, "w");

    pack_header(fp, NULL);
    fflush(fp);

    while ((pid = fork()) == -1);
    if (pid == 0) {
        dup2(fd, 1);
        dup2(fd, 2);
        close(fd);

        execl(prog, prog, NULL);
        perror(prog);
        exit(-1);
    }

    waitpid(pid, NULL, 0);
    return 0;
}

int resp_do_cat(char *f, int fd)
{
    int c;
    char *extension = get_filename_extension(f);
    char *content = "text/plain";
    FILE *fpsock, *fpfile;

    if (strcmp(extension, "html") == 0) {
        content = "text/html";
    } else if (strcmp(extension, "gif") == 0) {
        content = "text/gif";
    } else if (strcmp(extension, "jpg") == 0) {
        content = "text/jpg";
    } else if (strcmp(extension, "jpeg") == 0) {
        content = "text/jpeg";
    }

    fpsock = fdopen(fd, "w");
    fpfile = fopen(f, "r");
    if (fpsock != NULL && fpfile != NULL) {

```

```

        pack_header(fpsock, content);
        fprintf(fpsock, "\r\n");

        while ((c = getc(fpfile)) != EOF) {
            putc(c, fpsock);
        }
        fclose(fpfile);
        fclose(fpsock);
        return 0;
    }

    return 1;
}

void pack_header(FILE *fp, char *extension)
{
    fprintf(fp, "HTTP/1.1 200 OK\r\n");
    if (extension)
        fprintf(fp, "Content-type: %s\r\n", extension);
}

char *get_filename_extension(char *f)
{
    return strchr(f, '.') + 1;
}

void ignore_others(FILE *fp)
{
    char buf[BUFSIZE];

    fgets(buf, BUFSIZE, fp);
    while(strcmp(buf, "\r\n"))
        fgets(buf, BUFSIZE, fp);
    return;
}

```

### 附录 3. I/O 多路复用方式的 Web 服务器

```

/* web_server_select.c */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define BUFSIZE 1024
#define MAX_QUE_CONN_NM 10

typedef struct {
    int maxfd;
    int maxi;
    int nready;
    int clientfd[FD_SETSIZE];
    int len[FD_SETSIZE];
    char request[FD_SETSIZE][BUFSIZE];
    fd_set read_set;
    fd_set ready_set;
} pool;

int start_server(int portnum);
void init_pool(int sockfd, pool *p);
void add_client(int connfd, pool *p);
void check_clients(pool *p);

int process_request(char *rq, int fd);
int resp_cannot_do(int fd);
int not_exist(char *f);
int resp_do_404(char *item, int fd);
int is_a_dir(char *f);
int resp_do_ls(char *dir, int fd);
int is_a_cgi_file(char *f);
int resp_do_exec(char *prog, int fd);
int resp_do_cat(char *f, int fd);
void pack_header(FILE *fp, char *extension);
char *get_filename_extension(char *f);

int main(int argc, char *argv[])
{
    int sock;
    int fd;
    int portnum;

```

```

static pool pool;

if (argc != 2) {
    fprintf(stderr, "usage: %s <portnum>\n", argv[0]);
    exit(-1);
}
portnum = atoi(argv[1]);
sock = start_server(portnum);

init_pool(sock, &pool);

while(1) {
    pool.ready_set = pool.read_set;
    pool.nready = select(pool.maxfd + 1, &pool.ready_set, NULL, NULL, NULL);
    if ( pool.nready== -1) {
        perror("select");
        exit(-1);
    }

    if (FD_ISSET(sock, &pool.ready_set)) {
        if ((fd = accept(sock, NULL, NULL)) == -1) {
            perror("accept");
            close(sock);
            exit(-1);
        }
        add_client(fd, &pool);
    }

    check_clients(&pool);
}

}

int start_server(int portnum)
{
    struct sockaddr_in server_sockaddr;
    int sockfd;
    int i = 1;

    /* create socket link */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(-1);
    }
}

```



```

server_sockaddr.sin_family = AF_INET;
server_sockaddr.sin_port = htons(portnum);
server_sockaddr.sin_addr.s_addr = INADDR_ANY;
bzero(server_sockaddr.sin_zero, 8);

setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &i, sizeof(i));

if (bind(sockfd, (struct sockaddr *)&server_sockaddr, sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(-1);
}

if (listen(sockfd, MAX_QUE_CONN_NM) == -1)
{
    perror("listen");
    exit(-1);
}

return sockfd;
}

void init_pool(int sockfd, pool *p)
{
    int i;
    p->maxi = -1;
    for(i = 0; i < FD_SETSIZE; i++)
        p->clientfd[i] = -1;

    p->maxfd = sockfd;
    FD_ZERO(&p->read_set);
    FD_SET(sockfd, &p->read_set);
}

void add_client(int connfd, pool *p)
{
    int i;
    p->nready--;
    for (i = 0; i < FD_SETSIZE; i++)
        if (p->clientfd[i] < 0) {
            p->clientfd[i] = connfd;
            memset(p->request[i], 0, BUFSIZE);
            p->len[i] = 0;

            FD_SET(connfd, &p->read_set);
        }
}

```

```

        if (connfd > p->maxfd)
            p->maxfd = connfd;
        if (i > p->maxi)
            p->maxi = i;

        break;
    }
    if (i == FD_SETSIZE)
        printf("add_client error: Too many clients\n");
}

void check_clients(pool *p)
{
    int i;
    int connfd;
    int n;
    int ret;
    char buf[BUFSIZE];
    char *q;

    for (i=0; (i<=p->maxi)&&(p->nready>0); i++) {
        connfd = p->clientfd[i];

        if ((connfd > 0) && (FD_ISSET(connfd, &p->ready_set))) {
            p->nready--;
            ret = read(connfd, p->request[i]+p->len[i], BUFSIZE);
            if (ret == -1) {
                perror("read error");
                close(connfd);
                continue;
            } else if (ret == 0) {
                printf("close\n");
                if (close(connfd) < 0) {
                    perror("close connfd failed");
                    exit(-1);
                }
                FD_CLR(connfd, &p->read_set);
                p->clientfd[i] = -1;
            } else {
                p->len[i] += ret;
                q = &p->request[i][p->len[i]-4];
                if (strcmp(q, "\r\n\r\n") == 0) { //request[i] end
                    q = strchr(p->request[i], '\r');
                }
            }
        }
    }
}

```

```

        *q = '\0';
        process_request(p->request[i], connfd);
        close(connfd);
        FD_CLR(connfd, &p->read_set);
        p->clientfd[i] = -1;
    } // if(strcmp(q, "\r\n\r\n") == 0)
} // ret
} // if((connfd > 0) && (FD_ISSET(connfd, &p->ready_set)))
} // for
}

```

```

int process_request(char *rq, int fd)
{
    char cmd[BUFSIZE], arg[BUFSIZE];

    strcpy(arg, ".");
    if (sscanf(rq, "%s %s", cmd, arg + 2) != 2) {
        return 1;
    }

    if (strcmp(cmd, "GET") != 0)
    {
        resp_cannot_do(fd);
    } else if (not_exist(arg)) {
        resp_do_404(arg, fd);
    } else if (is_a_dir(arg)) {
        resp_do_ls(arg, fd);
    } else if (is_a_cgi_file(arg)) {
        resp_do_exec(arg, fd);
    } else {
        resp_do_cat(arg, fd);
    }

    return 0;
}

```

```

int resp_cannot_do(int fd)
{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 501 Not Implemented\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "That command is not yet implemented\r\n");
    fclose(fp);
}

```

```

        return 0;
    }

int not_exist(char *f)
{
    struct stat info;
    return (stat(f, &info) == -1);
}

int resp_do_404(char *item, int fd)
{
    FILE *fp = fdopen(fd, "w");
    fprintf(fp, "HTTP/1.1 404 Not Found\r\n");
    fprintf(fp, "Content-type: text/plain\r\n");
    fprintf(fp, "\r\n");
    fprintf(fp, "The item you requested: %s\r\nis not found\r\n", item);
    fclose(fp);
    return 0;
}

int is_a_dir(char *f)
{
    struct stat info;
    return (stat(f, &info) != -1 && S_ISDIR(info.st_mode));
}

int resp_do_ls(char *dir, int fd)
{
    pid_t pid;
    FILE *fp = fdopen(fd, "w");

    pack_header(fp, "text/plain");
    fprintf(fp, "\r\n");
    fflush(fp);

    while ((pid = fork()) == -1);
    if (pid == 0) {
        dup2(fd, 1);
        dup2(fd, 2);
        close(fd);

        execlp("ls", "ls", "-l", dir, NULL);
        perror(dir);
        exit(-1);
    }
}

```

```

    }

    waitpid(pid, NULL, 0);
    return 0;
}

int is_a_cgi_file(char *f)
{
    return(strcmp(get_filename_extension(f), "cgi") == 0);
}

int resp_do_exec(char *prog, int fd)
{
    pid_t pid;
    FILE *fp = fdopen(fd, "w");

    pack_header(fp, NULL);
    fflush(fp);

    while ((pid = fork()) == -1);
    if (pid == 0) {
        dup2(fd, 1);
        dup2(fd, 2);
        close(fd);

        execl(prog, prog, NULL);
        perror(prog);
        exit(-1);
    }

    waitpid(pid, NULL, 0);
    return 0;
}

int resp_do_cat(char *f, int fd)
{
    int c;
    char *extension = get_filename_extension(f);
    char *content = "text/plain";
    FILE *fpsock, *fpfile;

    if (strcmp(extension, "html") == 0) {
        content = "text/html";
    } else if (strcmp(extension, "gif") == 0) {

```

```

        content = "text/gif";
    } else if (strcmp(extension, "jpg") == 0) {
        content = "text/jpg";
    } else if (strcmp(extension, "jpeg") == 0) {
        content = "text/jpeg";
    }

    fpsock = fdopen(fd, "w");
    fpfile = fopen(f, "r");
    if (fpsock != NULL && fpfile != NULL) {
        pack_header(fpsock, content);
        fprintf(fpsock, "\r\n");

        while ((c = getc(fpfile)) != EOF) {
            putc(c, fpsock);
        }
        fclose(fpfile);
        fclose(fpsock);
        return 0;
    }
    return 1;
}

void pack_header(FILE *fp, char *extension)
{
    fprintf(fp, "HTTP/1.1 200 OK\r\n");
    if (extension)
        fprintf(fp, "Content-type: %s\r\n", extension);
}

char *get_filename_extension(char *f)
{
    return strchr(f, '.') + 1;
}

```

# 实验三 用 Qt 编写嵌入式 GUI 程序

## 一. 实验目的

1. 掌握利用 QtCreator 集成开发环境构建 Qt 工程、设计 GUI 界面、编写、编译 Qt 程序的基本方法。
2. 了解 Qt 程序框架和简单的 Qt 程序设计方法。
3. 了解 Qt 的信号和槽机制。

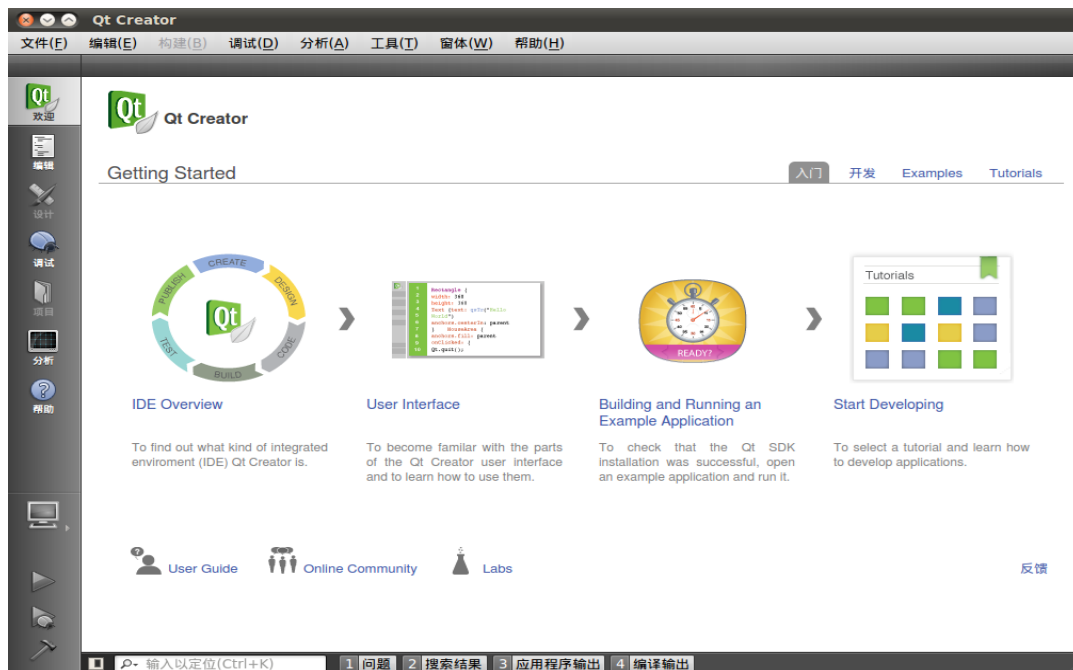
## 二. 实验内容

1. 查看 API 手册，学习简单的 Qt 类的使用，如 QLineEdit、QPushButton 等。
2. 用 QtCreator 创建工程，用 Qt 编写计算器程序。
3. 将计算器程序移植到实验箱。

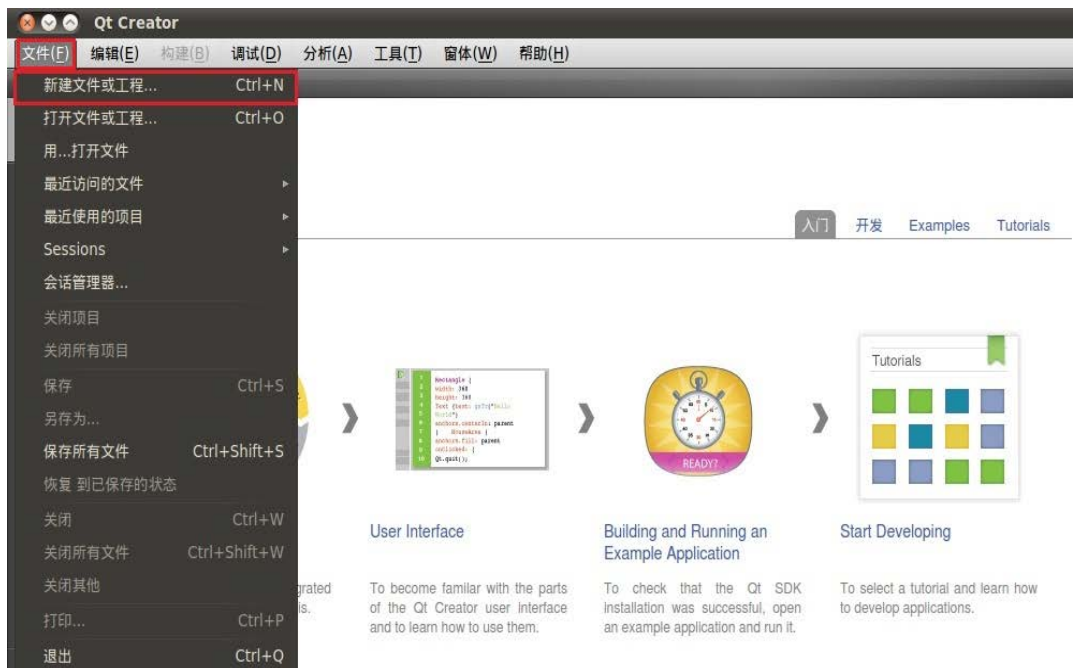
## 三. 实验步骤

### 3.1 创建 Qt 工程

1. 通过 Ubuntu 虚拟机桌面上的快捷方式打开 QtCreator。

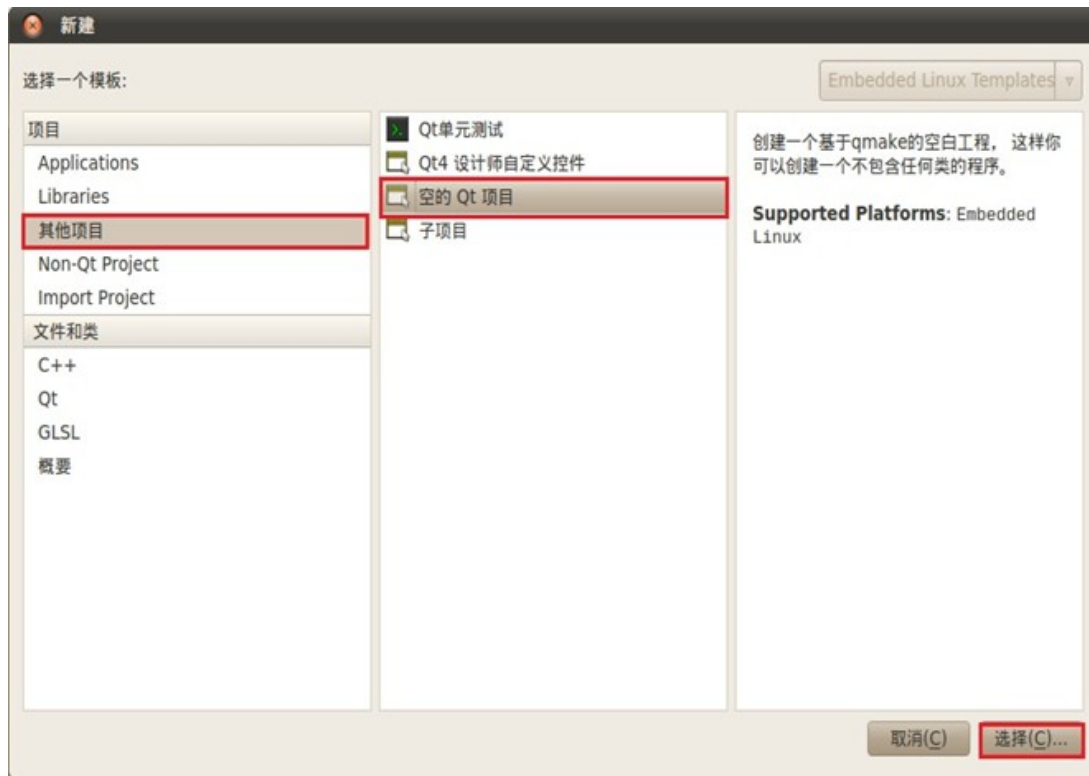


2. 新建工程，点击“文件（F）->新建文件或工程”。如下图所示。



3. 在弹出的窗口选择“其它项目->空的 Qt 项目”然后点击“选择”：





4. 定义工程名，填写“Calculator”，路径填“/opt/work/”然后设为默认路径并点“下一步”



5. 在目标设置窗口，在所看到的 Embedded linux 左边点击“详情”。把发布和调试目录设为 /opt/work/Calculator-build. 如下图。



6. 下一步点完成就可以了。

## 3.2 创建计算器程序

计算器程序主要分以下两部分工作：一是实现计算器的图形界面；二是实现按键事件和该事件对应的功能绑定，即信号和对应处理槽函数的绑定。

### 1. 计算器图形界面的实现

通过分析计算器的功能我们可知，需要 16 个按键和一个显示框，同时考虑到整体的排布，还需要水平布局器和垂直布局器。通过组织这些类我们可以实现一个简单的带有数字 0~9，可以进行简单四则运算且具有清屏功能的计算器。对于这些类的具体操作会在后面的代码中详细说明。

### 2. 信号和对应槽函数的绑定

分析计算器的按键我们可以把按键事件分为以下三类，一是简单的数字按键，主要进行数字的录入，这类按键包括按键 0~9；二是运算操作键，用于输入数学运算符号，进行数学运算和结果的显示，这类按键包括“+”，“-”，“\*”，“/”，“=”；三是清屏操作键，用于显示框显示信息的清除。

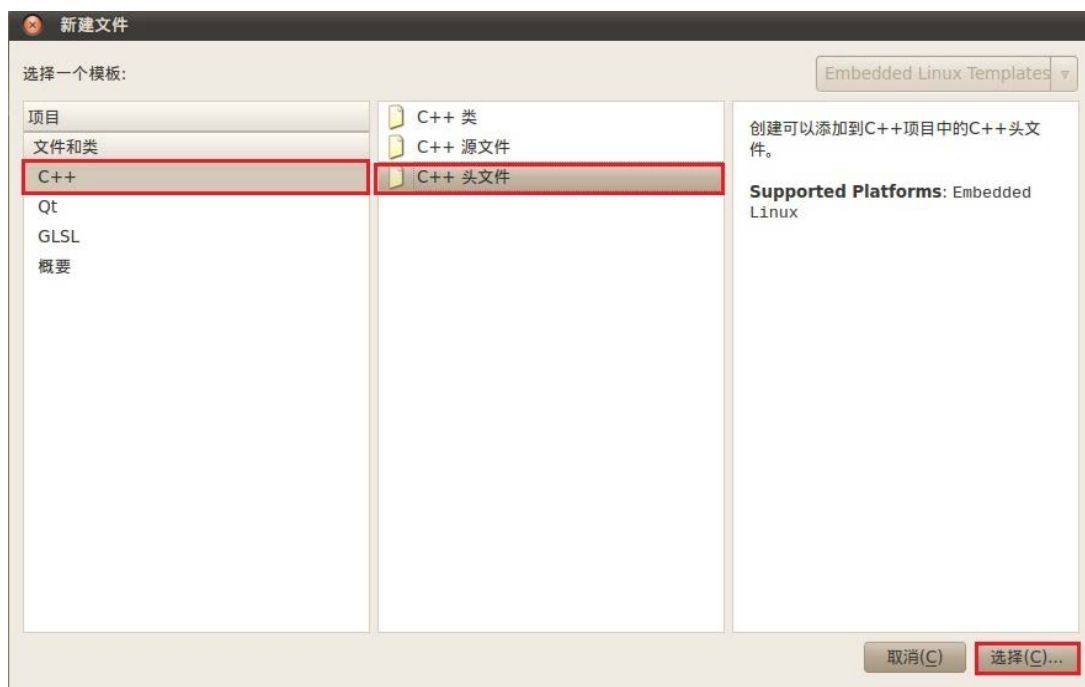
进入刚才创建的空工程，双击左侧的 Calculator.pro，在主编辑框中目前显示 Calculator.pro 的内容为空。

### 3. 添加文件 calculator.h（源程序见附录 1）

在工程 Calculator 上面点击右键，然后点击“添加新文件”



选择添加 C++ 头文件

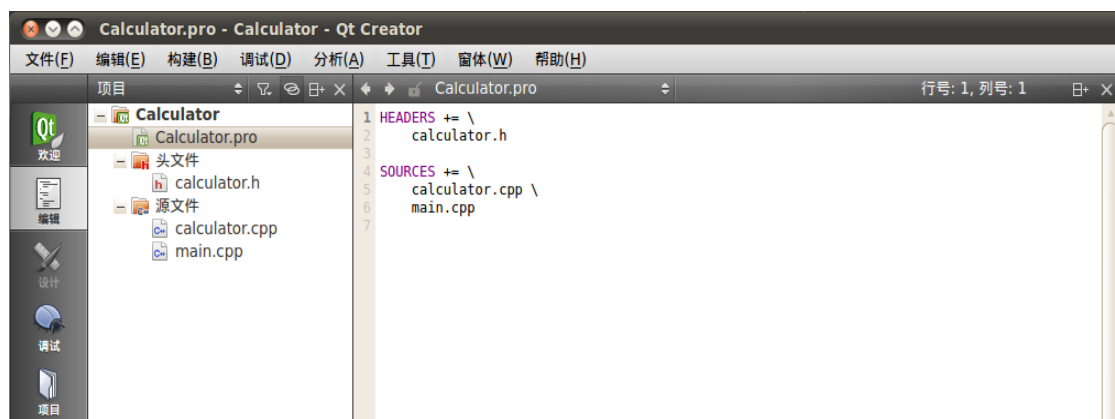


进入下一步后输入文件名 calculator.h（如图所示），然后完成文件的添加。



4. 添加文件 `calculator.cpp` 和 `main.cpp`（源程序见附录 2 和附录 3）

与添加文件 `calculator.h` 的过程类似，只是在选择文件类型时选择为 C++ 源文件。完成后可以查看 `Calculator.pro` 文件的内容。工程结构如下。



### 3.3 交叉编译 Qt 工程，并移植到目标机运行

1. 完成源程序的编辑后，从菜单点击“构建->构建项目 Calculator”（如下图所示），这时 QtCreator 会自动编译源程序并生成可执行程序（这里默认的编译环境是针对目标机的）。



可以在/opt/work/Calculator-build 目录下找到可执行程序。

```
root@ubuntu:/# cd /opt/work/Calculator-build/
root@ubuntu:/opt/work/Calculator-build# ls
Calculator calculator.o main.o Makefile moc_calculator.cpp moc_calculator.o
root@ubuntu:/opt/work/Calculator-build#
```

2. 用 scp 命令来发到目标机/root 目录。

```
root@ubuntu:/opt/work/Calculator-build# scp Calculator root@192.168.0.101:/root
root@192.168.0.101's password:
Calculator                               100%  40KB  40.5KB/s   00:00
root@ubuntu:/opt/work/Calculator-build#
```

3. 现在就可以在目标机上运行计算器程序。

```
[root@s210x ~]# cd /root
[root@s210x /root]# ls
Calculator
[root@s210x /root]# ./Calculator -qws&
[root@s210x /root]#
```

提示：实验箱的触摸屏暂时不可用，可以找一个 USB 鼠标插入到实验箱的 USB1~3 口，然后通过鼠标操作 Qt 界面。

## 四. 附录

### 附录 1 calculator.h 源代码

```
#ifndef CALCULATOR_H
#define CALCULATOR_H //对 calculator.h 头文件的声明
#include <QWidget> //包含主窗体类
#include <QPushButton> //包含按键类
#include <QVBoxLayout> //包含垂直布局器类
#include <QHBoxLayout> //包含水平布局器类
#include <QLineEdit> //包含显示框类

class Calculator : public QWidget //计算器继承自主窗体类
{
    Q_OBJECT //必须加上这句，如果要调用信号，槽函数的操作的话
public:
    Calculator(); //计算器类的构造函数
    ~Calculator(); //计算器类的析构函数

public slots: //定义各个按键按下后对应操作处理的槽函数
    void zeroButtonPress();
    void oneButtonPress();
    void twoButtonPress();
    void threeButtonPress();
    void fourButtonPress();
    void fiveButtonPress();
    void sixButtonPress();
    void sevenButtonPress();
    void eightButtonPress();
    void nineButtonPress();
    void addButtonPress();
    void subButtonPress();
    void mulButtonPress();
    void divButtonPress();
    void clearButtonPress();
    void equButtonPress();

private:
    QLineEdit *operateEdit; //声明显示框

    QPushButton *zeroButton; //声明数字按键1
    QPushButton *oneButton;
```

```

QPushButton *twoButton;
QPushButton *threeButton;
QPushButton *fourButton;
QPushButton *fiveButton;
QPushButton *sixButton;
QPushButton *sevenButton;
QPushButton *eightButton;
QPushButton *nineButton;

QPushButton *clearButton;//声明运算符按键
QPushButton *addButton;
QPushButton *subButton;
QPushButton *divButton;
QPushButton *mulButton;
QPushButton *equButton;

QHBoxLayout *firstLayout;//声明水平布局器，该布局器主要对 16 个按键进行布局
QHBoxLayout *secondLayout;
QHBoxLayout *thirdLayout;
QHBoxLayout *fourthLayout;

QVBoxLayout *mainLayout;//声明垂直布局器，该布局器主要对主窗体上面的空间进行
排布

QString input1;//计算器第一个运算操作数
QString input2;//计算器第二个运算操作数
char operate;//运算符
};
#endif // CALCULATOR_H

```

## 附录 2：calculator.cpp 源代码

首先是构造函数的实现：

```

Calculator::Calculator()
{
    operateEdit = new QLineEdit(this);//初始化显示框
    operateEdit->setReadOnly(true); //设置显示框为只读
    operateEdit->setText(tr("0"));//初始化显示框显示数据为0

    zeroButton = new QPushButton;//初始化按键
    zeroButton->setText(tr("0"));//设置按键上显示的标签，以下对按键相关的操作类似
    oneButton = new QPushButton;

```

```

oneButton->setText(tr("1"));
twoButton = new QPushButton;
twoButton->setText(tr("2"));
threeButton = new QPushButton;
threeButton->setText(tr("3"));
fourButton = new QPushButton;
fourButton->setText(tr("4"));
fiveButton = new QPushButton;
fiveButton->setText(tr("5"));
sixButton = new QPushButton;
sixButton->setText(tr("6"));
sevenButton = new QPushButton;
sevenButton->setText(tr("7"));
eightButton = new QPushButton;
eightButton->setText(tr("8"));
nineButton = new QPushButton;
nineButton->setText(tr("9"));
clearButton = new QPushButton;
clearButton->setText(tr("Clear"));
addButton = new QPushButton;
addButton->setText(tr("+"));
subButton = new QPushButton;
subButton->setText(tr("-"));
mulButton = new QPushButton;
mulButton->setText(tr("*"));
divButton = new QPushButton;
divButton->setText(tr("/"));
equButton = new QPushButton;
equButton->setText(tr("="));

```

```

firstLayout = new QHBoxLayout; //初始化水平布局器 firstLayout
firstLayout->addWidget(zeroButton); //把按键 zeroButton 添加到 firstLayout
firstLayout->addWidget(oneButton); //把按键 oneButton 添加到 firstLayout
firstLayout->addWidget(twoButton); //把按键 twoButton 添加到 firstLayout
firstLayout->addWidget(addButton); //把按键 threeButton 添加到 firstLayout, 以下对水平
布局器的操作类似

```

```

secondLayout = new QHBoxLayout;
secondLayout->addWidget(threeButton);
secondLayout->addWidget(fourButton);
secondLayout->addWidget(fiveButton);
secondLayout->addWidget(subButton);

```

```

thirdLayout = new QHBoxLayout;

```



```

thirdLayout->addWidget(sixButton);
thirdLayout->addWidget(sevenButton);
thirdLayout->addWidget(eightButton);
thirdLayout->addWidget(mulButton);

fourthLayout = new QHBoxLayout;
fourthLayout->addWidget(nineButton);
fourthLayout->addWidget(clearButton);
fourthLayout->addWidget(equButton);
fourthLayout->addWidget(divButton);

mainLayout = new QVBoxLayout(this);//初始化垂直布局器 mainLayout
mainLayout->addWidget(operateEdit); //把显示数据框 operateEdit 添加到 mainLayout
mainLayout->addLayout(firstLayout); //把水平布局器 firstLayout 添加到 mainLayout
mainLayout->addLayout(secondLayout); //把水平布局器 secondLayout 添加到
mainLayout
mainLayout->addLayout(thirdLayout); //把水平布局器 thirdLayout 添加到 mainLayout
mainLayout->addLayout(fourthLayout); //把水平布局器 fourthLayout 添加到 mainLayout

connect(zeroButton,SIGNAL(clicked()),this,SLOT(zeroButtonPress()));
//把按键 zeroButton 的按下事件同 zeroButtonPress()绑定到一起，以下操作类似
connect(oneButton,SIGNAL(clicked()),this,SLOT(oneButtonPress()));
connect(twoButton,SIGNAL(clicked()),this,SLOT(twoButtonPress()));
connect(threeButton,SIGNAL(clicked()),this,SLOT(threeButtonPress()));
connect(fourButton,SIGNAL(clicked()),this,SLOT(fourButtonPress()));
connect(fiveButton,SIGNAL(clicked()),this,SLOT(fiveButtonPress()));
connect(sixButton,SIGNAL(clicked()),this,SLOT(sixButtonPress()));
connect(sevenButton,SIGNAL(clicked()),this,SLOT(sevenButtonPress()));
connect(eightButton,SIGNAL(clicked()),this,SLOT(eightButtonPress()));
connect(nineButton,SIGNAL(clicked()),this,SLOT(nineButtonPress()));

connect(addButton,SIGNAL(clicked()),this,SLOT(addButtonPress()));
connect(subButton,SIGNAL(clicked()),this,SLOT(subButtonPress()));
connect(mulButton,SIGNAL(clicked()),this,SLOT(mulButtonPress()));
connect(divButton,SIGNAL(clicked()),this,SLOT(divButtonPress()));
connect(equButton,SIGNAL(clicked()),this,SLOT(equButtonPress()));
connect(clearButton,SIGNAL(clicked()),this,SLOT(clearButtonPress()));

this->setWindowTitle(tr("Calculator"));//设置窗体标题为 Calculator

input2= "0";//初始化运算操作数2为0
input1 = "0";//初始化运算操作数1为0
operate = '0';//初始化运算符为'0'
}

```

然后是析构函数的实现：

Calculator::~Calculator()//析构函数主要完成对构造函数中所声明的 QLineEdit、QPushButton、QHBoxLayout、QVBoxLayout 类的对象的回收工作（可以不定义析构函数，程序运行结束时会自动调用系统默认的析构函数）

```
{
    if (operateEdit != NULL) //
    {
        delete operateEdit;
        operateEdit = NULL;
    }

    if (zeroButton != NULL)
    {
        delete zeroButton;
        zeroButton = NULL;
    }
    .....
}
```

根据前面对按键事件的分析，有数字输入键，运算操作符输入键和清屏键三种，故对每种事件的槽响应函数都只说明一种，其他依此类推。

数字输入键响应槽函数，以按键“1”为例：

```
void Calculator::oneButtonPress()
{
    if(input2=="0")//如果当前显示框为0
    {
        input2="1";//变0为1
    }
    else//如果当前显示框不为0
    {
        input2= operateEdit->text();
        input2.append(tr("1"));//在显示的数据后面追加1
    }
    operateEdit->setText(input2);//更新显示框中的显示信息
}
```

运算操作符输入键响应槽函数，以按键“+”为例：

```
void Calculator::addButtonPress()
{
    float first,second;
    input2= operateEdit->text();//把当前显示的数据保存到运算操作数2中
    if(operate == '0')//如果是第一次按下运算符键
    {
        input1 = input2;//把运算操作数2中的数据保存到运算操作数1中
        input2 = "0";//清除运算操作数2中的数据
    }
}
```

```

        operate = '+';//把运算符键置‘+’
    }
    Else//如果是第二次按下运算符键
    {
        second=input2.toFloat();//把运算操作数2中的数据转化为浮点类型
        first=input1.toFloat();//把运算操作数1中的数据转化为浮点类型
        switch(operate)//根据当前的运算符判断做何操作
        {
            case '+':first = first+second;break;
            case '-':first = first-second;break;
            case '*':first = first*second;break;
            case '/':first = first/second;break;
        }
        input1 = QString::number(first,'f',10);//把运算的结果转化成为可以在显示框显示的
类型
        input2 = "0";//清除运算操作数2中的数据
        operate = '+';//把运算符键置‘+’
    }
    operateEdit->setText(input1);//更新显示框中的显示内容
}

```

清屏操作响应函数：

```

void Calculator::clearButtonPress()
{
    operate = '0'; //把运算符键置‘0’
    input2 = "0";//把运算操作数2清零
    input1 = "0";//把运算操作数1清零
    operateEdit->setText(input2); //更新显示框中的显示内容
}

```

## 附录 3： main.cpp 源代码

```

#include <QApplication>//包含应用程序类
#include "calculator.h"//包含计算器类

int main(int argc, char *argv[])//main 函数的标准写法
{
    QApplication app(argc, argv); //创建一个 QApplication 对象，管理应用程序的资源
    Calculator mainwindow; //产生一个计算器对象
    mainwindow.showMaximized();//显示计算器窗体(默认以最大化的形式显示)
    return app.exec();//让程序进入消息循环，等待可能的菜单、工具条、鼠标等的输入，
进行响应。
}

```



# 实验四 编写嵌入式 Linux 设备驱动程序

## 一. 实验目的

1. 掌握简单的 Linux 字符设备驱动程序的设计方法。
2. 掌握在 Linux 应用程序中访问设备驱动的方法，以及应用程序与驱动程序的通信调用机制
3. 掌握嵌入式 linux 下字符驱动程序的编译和运行方法。
4. 掌握嵌入式 linux 字符驱动模块的动态加载，卸载；

## 二. 实验内容

1. 编译 Linux 内核，构建驱动程序的交叉编译环境。
2. 交叉编译驱动程序和对应的应用程序。
3. 将驱动程序和应用程序下载到目标机中运行。

## 三. 试验步骤

### 3.1 编译 Linux 内核

Linux 内核源码就在虚拟机的/opt/cross-compiler/kernel-embv210 目录下。

在 Ubuntu 虚拟机中打开一个终端，采用如下命令编译该 Linux 内核源码。

```
$ cd /opt/cross-compiler/kernel-embv210
```

```
$ make distclean
```

```
$ cp embv210.config .config
```

```
$ make zImage
```

注意 zImage 的大小写！

## 3.2 编译驱动程序和应用程序

1. 把所附录中提供的驱动程序源代码和应用程序源代码分别拷贝到 /opt/work/hello\_driver 和 /opt/work/hello\_test 目录下。
2. 进入到 hello\_test 目录。用 arm-none-linux-gnueabi-gcc 来编译应用程序。然后把结果拷贝到 /opt/tftp 目录。如下图。

```
root@ubuntu:/opt/work/hello_test# arm-none-linux-gnueabi-gcc -o hello_test hello_test.c
root@ubuntu:/opt/work/hello_test# ls
hello_test  hello_test.c  Makefile
root@ubuntu:/opt/work/hello_test# cp hello_test /opt/tftp
```

3. 为了编译 hello\_driver 驱动程序。我们将使用 make 工具和 Makefile 文件。先进入 /opt/work/hello\_driver, 然后用 gedit 编辑器来查看 Makefile 文件, 保证它所指向的内核路径是否正确。内核路径就是 /opt/cross-compiler/kernel-embv210。编辑完之后保存并关闭。

```
INSTALLDIR      = /opt/tftp

ifneq ($(KERNELRELEASE),)
obj-m:=hello_driver.o
else
KERNELDIR:=/opt/cross-compiler/kernel-embv210
PWD:=$(shell pwd)

default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
    rm -rf *.o *.order *.cmd *.ko *.mod.c *.symvers
endif

install: hello_driver.ko
    mkdir -p $(INSTALLDIR)
    cp --target-dir=$(INSTALLDIR) hello_driver.ko
```

INSTALLDIR 是编译结果的安装目录, 运行“make install”命令就可以将生成的可执行程序拷贝到 /opt/tftp。

4. 在/opt/work/hello\_driver 目录, 用 make 命令编译驱动程序。Make 工具会执行 Makefile 文件中的 default 部分。可以看到如下运行结果。

```
root@ubuntu:/opt/work# cd hello_driver/
root@ubuntu:/opt/work/hello_driver# make
make -C /opt/cross-compiler/kernel-embv210 M=/opt/work/hello_driver modules
make[1]: 正在进入目录 `/opt/cross-compiler/kernel-embv210'
  CC [M] /opt/work/hello_driver/hello_driver.o
/opt/work/hello_driver/hello_driver.c: In function 'hello_read':
/opt/work/hello_driver/hello_driver.c:51: warning: ignoring return value of 'copy_to_user', declared with attribute warn_unused_result
/opt/work/hello_driver/hello_driver.c: At top level:
/opt/work/hello_driver/hello_driver.c:76: warning: initialization from incompatible pointer type
/opt/work/hello_driver/hello_driver.c: In function 'hello_exit':
/opt/work/hello_driver/hello_driver.c:146: warning: passing argument 2 of 'device_destroy' makes integer from pointer without a cast
include/linux/device.h:601: note: expected 'dev_t' but argument is of type 'char *'
  Building modules, stage 2.
  MODPOST 1 modules
  CC /opt/work/hello_driver/hello_driver.mod.o
  LD [M] /opt/work/hello_driver/hello_driver.ko
make[1]: 正在离开目录 `/opt/cross-compiler/kernel-embv210'
root@ubuntu:/opt/work/hello_driver#
```

5. 现在用 make install 命令将 hello\_driver.ko 拷贝到安装目录。

```
root@ubuntu:/opt/work/hello_driver# make install
mkdir -p /opt/tftp
cp --target-dir=/opt/tftp hello_driver.ko
```

6. 现在用 make clean 来删除所有后来不用的文件。Make 工具会执行 Makefile 文件中的 clean 部分。
7. 确认宿主机和目标机的串口线和网线是否连好。打开目标机电源, 并在宿主机终端用 minicom 命令打开串口软件。
8. 用 tftp 服务把 hello\_test 和 hello\_driver.ko 文件考到目标机上去。

```
[root@s210x ~]#mkdir -p /mnt/example
[root@s210x ~]#cd /mnt/example
[root@s210x /mnt/example]#tftp -g 192.168.0.7 -r ./hello_test -l ./hello_test
[root@s210x /mnt/example]#tftp -g 192.168.0.7 -r ./hello_driver.ko -l ./hello_driver.ko
[root@s210x /mnt/example]#ls
hello_driver.ko  hello_test
```

9. 修改下载文件的可执行权限。

```
[root@s210x /mnt/example]#chmod -R 777 hello_driver.ko
[root@s210x /mnt/example]#chmod -R 777 hello_test
```

## 10. 用 insmod 命令加载驱动 hello\_driver.ko

```
[root@s210x /mnt/example]#insmod hello_driver.ko
[ 486.840059] The driver is insmoded successfully.
```

## 11. 执行 hello\_test，结果如下图。

```
[root@s210x /mnt/example]#./hello_test
[ 707.745499] open the description successfully.
value=1:
value=2:
value=3:
value=4:
value=5:
value=6:
value=7:
value=8:
value=9:
[ 716.746388] close the description successfully.
[root@s210x /mnt/example]#
```

## 12. 还可以查看并卸载已经加载的驱动。

```
[root@s210x /mnt/example]#lsmod
Module                Size  Used by    Not tainted
hello_driver          1793  0
[root@s210x /mnt/example]#rmmod hello_driver.ko
[ 783.258964] The driver is rmmoded successfully.
```

# 四. 附录

## 附录 1 hello\_test 源码

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define DEVICE_NAME "/dev/hello"

int main(int argc, char **argv)
{
```



```

int fd,i;
unsigned char val=0,value;
fd=open("/dev/hello",O_RDWR);
if(fd==-1){
    printf("Failed to open device %s.\n",DEVICE_NAME);
    return -1;
}
for(i=0;i<9;i++)
{
    val=val+1;
    write(fd,&val,sizeof(val));
    if(read(fd,&value,sizeof(value))!=0)
    {
        printf("value=%0x:\n",value);
    }
    sleep(1);
}
close(fd);
return 0;
}

```

## 附录 2 hello\_driver 源码

```

#include<linux/init.h>
#include<linux/module.h>
#include<linux/sched.h>
#include<linux/kernel.h>
#include <asm/uaccess.h>
#include<linux/device.h>
#include <linux/cdev.h>
#include <linux/fs.h>
/*hello driver structure*/
#define HELLO_DEVICE   "hello_test"
#define HELLO_NODE     "hello"
/*declare device number variable*/
static dev_t num_dev;
/*declare character driver variable*/
static struct cdev *cdev_p;
/*declare a class*/
static struct class *hello_class;
/*declare and initialize a variable*/
static unsigned char hello_value = 0;
static int hello_open(struct inode* inode,struct file* filp)
{

```

```

        /* adds the number of times, the owner of this device can manage it */
        printk(KERN_INFO"open the description successfully.\n");
        try_module_get(THIS_MODULE);
        return 0;
    }
static int hello_release(struct inode* inode,struct file* filp)
{
    /*reduces the number of times, the owner of this device can manage it*/
    printk(KERN_INFO"close the description successfully.\n");
    module_put(THIS_MODULE);
    return 0;
}
static ssize_t hello_read(struct file* filp,char __user *buf,size_t count,loff_t* f_pos)
{
    copy_to_user(buf, (char *)&hello_value, sizeof(unsigned char));
    return sizeof(unsigned char);
}
static ssize_t hello_write(struct file* filp,char __user *buf,size_t count,loff_t* f_pos)
{
    unsigned char value;
    if(count==1)
    {
        /*write data on the user space, if this fails then return error*/
        if(copy_from_user(&value, buf,sizeof(unsigned char)))
            return -EFAULT;
        hello_value=(value&0x0F);
        return sizeof(unsigned char);
    }else
        return -EFAULT;
}
/*declare file operations*/
static struct file_operations hello_fops={
    .owner      = THIS_MODULE,
    .open       = hello_open,
    .release    = hello_release,
    .read       = hello_read,
    .write      = hello_write,
};
/*Initialize the LED lights and load the LED device driver*/
static int hello_ctrl_init(void)
{
    int err;
    struct device* temp=NULL;
    /* dynamically delete hello_test device, num_dev is the device id */

```

```

err=alloc_chrdev_region(&num_dev,0,1,HELLO_DEVICE);
if (err < 0) {
    printk(KERN_ERR "HELLO: unable to get device name %d/n", err);
    return err;
}
/*dynamically allocate cdev RAM space*/
cdev_p = cdev_alloc();
cdev_p->ops = &hello_fops;
/*load the device driver*/
err=cdev_add(cdev_p,num_dev,1);
if(err){
    printk(KERN_ERR "HELLO: unable to add the device %d/n", err);
    return err;
}
/* create hello_test folder at the /sys/class directory*/
hello_class=class_create(THIS_MODULE,HELLO_DEVICE);
if(IS_ERR(hello_class))
{
    err=PTR_ERR(hello_class);
    goto unregister_cdev;
}
/* create hello device file based on /sys/class/hello_test and /dev */
temp=device_create(hello_class, NULL,num_dev, NULL, HELLO_NODE);
if(IS_ERR(temp))
{
    err=PTR_ERR(temp);
    goto unregister_class;
}
return 0;
unregister_class:
    class_destroy(hello_class);
unregister_cdev:
    cdev_del(cdev_p);
return err;
}
/*initialization*/
static int __init hello_init(void)
{
    int ret;
    printk("The driver is insmoded successfully.\n");
    ret = hello_ctrl_init();
    if(ret)
    {
        printk(KERN_ERR "Apply: Hello_driver_init--Fail !!!/n");
    }
}

```

```

        return ret;
    }
    return 0;
}
/*exit */
static void __exit hello_exit(void)
{
    printk("The driver is rmmoded successfully.\n");
    device_destroy(hello_class,HELLO_DEVICE);
    class_destroy(hello_class);
    cdev_del(cdev_p);
    unregister_chrdev_region(num_dev,1);
}
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("First Linux Driver");
module_init(hello_init);
module_exit(hello_exit);

```

## 附录 3 编译驱动程序的 Makefile 文件

```

INSTALLDIR = /opt/tftp
ifneq ($(KERNELRELEASE),)
obj-m:=hello_driver.o
else
KERNELDIR:=/usr/local/src/EMobile/EMBV210/kernel-embv210
PWD:=$(shell pwd)
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
    rm -rf *.o *.order *.cmd *.ko *.mod.c *.symvers
endif
install: hello_driver.ko
    mkdir -p $(INSTALLDIR)
    cp --target-dir=$(INSTALLDIR) hello_driver.ko

```

# 实验作业

## 作业 1

1. 编写 Linux 平台下的 C 语音程序，实现如下功能：

利用匿名管道实现两个兄弟进程（同一父进程创建的两个子进程）之间的通信。具体要求一个进程从键盘上循环接收用户输入的字符串，然后利用管道将字符串发送给另一个进程。接收者将收到的字符串打印到标准输出设备上。父进程等待两个子进程退出后自己再退出。

2. 在 PC 上的 Linux 系统中将上述程序调通之后，再交叉编译，将最终生成的可执行程序下载到实验箱中运行。

## 作业 2

编写一个 Hello World 版的 Qt GUI 程序（显示一个窗口，窗口中显示 Hello World 文字），交叉编译后再移植到实验箱中运行。

## 作业 3

将如下 LED 驱动程序和对应 LED 控制应用程序交叉编译，并将移植到实验箱中运行。交叉编译 LED 驱动程序所需的 Makefile 文件根据 hello\_driver 驱动的 Makefile 改写。

### 1. 驱动程序 led.c

```
/*头文件*/
#include<linux/init.h>
#include<linux/module.h>
#include<linux/sched.h>
#include<linux/kernel.h>
#include <asm/uaccess.h>
#include <plat/gpio-cfg.h>
#include <linux/gpio.h>
```

```

#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/device.h>

/*定义设备目录*/
#define DEVICE_LIST "led_test2"

/*定义设备文件节点*/
#define DEVICE_NODE "led_light2"

#define LED1 0x01
#define LED2 0x02
#define LED3 0x04
#define LED4 0x08

/*定义申请设备号(主设备号+次设备号)的变量*/
static dev_t num_dev;

/*字符设备的变量定义*/
static struct cdev *cdev_p;

/*定义一个 class 类*/
static struct class *led_dev_class;

/*定义一个全局变量，表示 LED 灯的状态*/
static unsigned char led_status = 0;

/*设置 LED 灯的状态*/
static void set_led_status(unsigned char status)
{
    /*表示 LED 灯的状态是否发生变化*/
    unsigned char led_status_changed;

    led_status_changed= led_status^(status & 0xF);

    /*数据变化检测*/
    led_status=(status & 0xF);

    /*如果 4 个 LED 灯的状态发生了变化*/
    if(led_status_changed!=0x00)
    {
        /*判断是否改变 LED1 灯的状态*/
        if(led_status_changed&LED1)
        {

```

```

        if(led_status&LED1)
            gpio_direction_output(S5PV210_GPH0(0),0);
        else
            gpio_direction_output(S5PV210_GPH0(0),1);
    }

    /*判断是否改变 LED2 灯的状态*/
    if(led_status_changed&LED2)
    {
        if(led_status&LED2)
            gpio_direction_output(S5PV210_GPH0(1),0);
        else
            gpio_direction_output(S5PV210_GPH0(1),1);
    }

    /*判断是否改变 LED3 灯的状态*/
    if(led_status_changed&LED3)
    {
        if(led_status&LED3)
            gpio_direction_output(S5PV210_GPH0(2),0);
        else
            gpio_direction_output(S5PV210_GPH0(2),1);
    }

    /*判断是否改变 LED4 灯的状态*/
    if(led_status_changed&LED4)
    {
        if(led_status&LED4)
            gpio_direction_output(S5PV210_GPH0(3),0);
        else
            gpio_direction_output(S5PV210_GPH0(3),1);
    }
}

}

/*读取 LED 灯的状态*/
static ssize_t s5pv210_led_read(struct file * file,char * buf,size_t count,loff_t * f_ops)
{
    /*从用户空间读取数据,获取 LED 灯的状态*/
    copy_to_user(buf, (char *)&led_status, sizeof(unsigned char));
    return sizeof(unsigned char);
}

```

```

/*定义实现 LED 灯的写操作*/
static ssize_t s5pv210_led_write (struct file * file,const char * buf, size_t count,loff_t * f_ops)
{
    unsigned char status;
    if(count==1)
    {
        /*向用户空间写数据,如果写失败, 则返回错误*/
        if(copy_from_user(&status, buf,sizeof(unsigned char)))
            return -EFAULT;
        set_led_status(status);
        return sizeof(unsigned char);
    }else
        return -EFAULT;
}

/*打开 LED 设备*/
static ssize_t s5pv210_led_open(struct inode * inode,struct file * file)
{
    /*增加管理此设备的 owner 模块的使用计数*/
    try_module_get(THIS_MODULE);
    return 0;
}

/*释放 LED 设备*/
static ssize_t s5pv210_led_release(struct inode * inode, struct file * file)
{
    /*减少管理此设备的 owner 模块的使用计数*/
    module_put(THIS_MODULE);
    return 0;
}

/*定义具体的文件操作*/
static const struct file_operations s5pv210_led_ctrl_ops={
    .owner          = THIS_MODULE,
    .open           = s5pv210_led_open,
    .read           = s5pv210_led_read,
    .write          = s5pv210_led_write,
    .release        = s5pv210_led_release,
};

/*LED 灯的初始化和 LED 设备驱动的加载*/
static int s5pv210_led_ctrl_init(void)
{
    int err;

```



```

struct device* temp=NULL;
unsigned int gpio;

/*GPIO 口的初始化 LED1,LED2,LED3,LED4, 设置为输出*/
for(gpio=S5PV210_GPH0(0);gpio<S5PV210_GPH0(4);gpio++)
{
    s3c_gpio_cfgpin(gpio, S3C_GPIO_SFN(1));
}

/*动态注册 led_test 设备,num_dev 为动态分配出来的设备号(主设备号+次设备号)*/
err=alloc_chrdev_region(&num_dev,0,1,DEVICE_LIST);
if (err < 0) {
    printk(KERN_ERR "LED: unable to get device name %d/n", err);
    return err;
}

/*动态分配 cdev 内存空间*/
cdev_p = cdev_alloc();
cdev_p->ops = &s5pv210_led_ctrl_ops;

/*加载设备驱动*/
err=cdev_add(cdev_p,num_dev,1);
if(err){
    printk(KERN_ERR "LED: unable to add the device %d/n", err);
    return err;
}

/*在/sys/class 下创建 led_test 目录*/
led_dev_class=class_create(THIS_MODULE,DEVICE_LIST);
if(IS_ERR(led_dev_class))
{
    err=PTR_ERR(led_dev_class);
    goto unregister_cdev;
}

/*基于/sys/class/led_test 和/dev 下面创建 led_light 设备文件*/
temp=device_create(led_dev_class, NULL,num_dev, NULL, DEVICE_NODE);
if(IS_ERR(temp))
{
    err=PTR_ERR(temp);
    goto unregister_class;
}

return 0;

```

```

unregister_class:
    class_destroy(led_dev_class);
unregister_cdev:
    cdev_del(cdev_p);
    return err;
}

/*模块的初始化*/
static int __init s5pv210_led_init(void)
{
    int ret;

    ret = s5pv210_led_ctrl_init();
    if(ret)
    {
        printk(KERN_ERR "Apply: S5PV210_LED_init--Fail !!!/n");
        return ret;
    }
    return 0;
}

/*模块的退出*/
static void __exit s5pv210_led_exit(void)
{
    device_destroy(led_dev_class,DEVICE_NODE);
    class_destroy(led_dev_class);
    cdev_del(cdev_p);
    unregister_chrdev_region(num_dev,1);
}

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("LED driver test");

module_init(s5pv210_led_init);
module_exit(s5pv210_led_exit);

```

## 2. LED 测试程序 led\_test.c:

```

// LED test programme
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define DEVICE_NODE "/dev/led_light2" //device point

```

```

int main(int argc,char **argv)
{
    int fd,i,j;
    unsigned char status;
    unsigned char t;

    /*打开设备节点*/
    fd = open(DEVICE_NODE,O_RDWR);
    if(fd == -1)
    {
        printf("open device %s error \n",DEVICE_NODE);
        return -1;
    }

    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            //依次点亮 LED1..LED4
            t=(unsigned char)((1<<j)&0x0F);
            write(fd,&t,sizeof(t));
            if(read(fd,&status,1)!=0)
            {
                printf("led status:%0x\n",status);
            }
            sleep(1);
        }
    }
    close(fd);
    return 0;
}

```