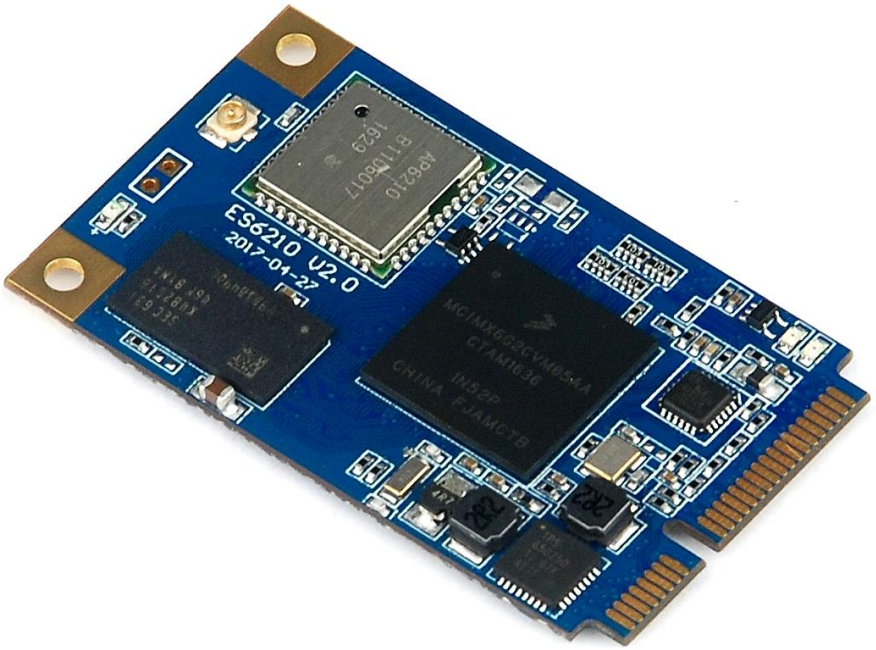
Emtronix



ES6210 工控主板使用必读

感谢您选择英创 ES6210 工控主板。

为了让您能够尽快地使用好我们的产品，英创公司编写了这篇《使用必读》，我们建议每一位使用英创产品的用户都浏览一遍。我们本着通俗易懂的原则，按照由浅入深的顺序，采用了大量图片和浅显的文字，以便于用户能边了解、边动手，轻松愉快地完成产品的开发。

在使用英创产品进行应用开发的过程中，如果您遇到任何困难需要帮助，都可以通过以下三种方式寻求英创工程师的技术支持：

1、直接致电 **028-86180660　85329360**

2、发送邮件到技术支持邮箱 **support@emlinix.com**

3、登录英创网站 **www.emlinix.com**，在技术论坛上直接提问

另，本手册以及其它相关技术文档、资料均可以通过英创网站下载。

注：英创公司将会不断完善本手册的相关技术内容，请客户适时从公司网站下载最新版本的手册，恕不另行通知。

再次谢谢您的支持！

1 ES6210 简介

感谢您购买英创信息技术有限公司的产品：ES6210 工控主板。

ES6210 采用 mini PCIe 全长卡的尺寸，使主板与底板的高度被控制在 5.5mm 之内。能更为方便地嵌入各种对外形尺寸敏感的工业设备之中。并且 ES6210 上搭载了蓝牙 Wi-Fi 模块 AP6210，能够很好的应用于 IoT（物联网）这一类的场合中。

ES6210 系列主板是面向工业领域的高性价比嵌入式主板，以 NXP 的 Cortex®-A7 芯片 iMX6UL 为其硬件核心，ES6210 通过预装完整的操作系统及接口驱动，为用户构造了可直接使用的通用嵌入式核心平台。目前 ES6210 可选择预装 Linux-4.1.15 系统平台，用户应用程序开发方面，可采用英创公司提供的 Eclipse 集成开发环境（Windows 版本），其编译生成的程序可直接运行与 ES6210。英创公司针对 ES6210 提供了完整的接口低层驱动以及丰富的应用程序范例，用户可在此基础上方便、快速地开发出各种工控产品。

ES6210 开发的基本文档包括：

《ES6210 工控主板使用必读》—— ES6210 快速入门手册，建议新客户都浏览一遍

《ES6210 工控主板数据手册》——ES6210 接口定义、电气特性以及各项技术指标

《ES6210 工控主板编程参考手册》——ES6210 功能接口使用方法及软件操作说明

《ES6210 评估底板数据手册》—— ES6210 的评估底板使用说明

ES6210 的更多资料和说明请参考 ES6210 开发光盘和登录我们的网站：http://www.emtronix.com/product/ ES6210.html。

2 搭建硬件开发平台

2.1 ES6210 开发评估套件说明

用户第一次使用 ES6210 往往是购买开发评估套件，开发评估套件包括如下几部分：

- **ES6210 工控主板一块：**NXP iMX6UL 处理器，预装嵌入式 Linux-4.1.15 实时多任务操作系统，接口资源丰富
- **ES6210 开发评估底板一块：**搭载 ES6210 并引出其板载资源。底板上提供了 ES6210 所有板载资源的标准接口，既方便用户对 ES6210 进行评估和开发，又为用户的外围硬件开发提供一定的参考
- **以太网连接线一条：**直连方式，用于进行目标机系统的管理维护以及开发网络方面的应用功能
- **USB 连接线一条：**AM 转 Mini-B 连接线，用于供电以及输出调试信息。
- **直流电源线一条：**红黑双色，+5V，用于为系统供电
- **开发资料光盘一张：**为用户的开发提供丰富翔实的软硬件资料

根据客户所开发的产品不同的需求，除了以上一些客户开发的必要配备外，客户可能还有一些其它开发附件，如：

- 各种尺寸的彩色显示屏，如 4.3 寸（480×272）、7 寸（800×480）等
- 英创公司提供的其它配套模块产品，如键盘扩展模块、AD 扩展模块等等
- GPRS/CDMA 通讯模块（如：Siemens MC37i R3）以及天线等附件
- 3G/4G 通讯模块（如：MU709s）以及天线等附件
- 客户所需要的其它附件

这些附件的配套使用方法，请参考该产品的使用说明或手册。

2.2 必要的准备

用户要利用 ES6210 进行开发，需要作如下一些必要准备：

- 准备一台带以太网接口、USB 接口的 PC 机作为开发主机，该 PC 机安装 Windows XP、Windows 7 操作系统或 Linux 操作系统。

  注：1、ES6210 的供电和调试口均由 PC 的 USB 接口实现，ES6210 的平均工作电流为 130mA，采用 PC 的 USB 口供电可完全足够其需求，调试口由底板上的串口转 USB 芯片转换而来，接入 PC 后会显示有新增加的终端，利用终端工具即可进行调试。

  2、如果用户在主机上使用 Linux 操作系统进行开发，由于 Linux 的开源和自由性，以及市面和互联网上已具备的丰富翔实的参考资料和各种 Linux 社区资源，英创公司不再对 Linux 环境下的开发过程进行技术支持，特此说明。

- 准备一台网络连接设备（集线器/交换机/路由器）。
- 准备一只可供临时存储数据的 U 盘。

2.3 开发环境的硬件连接和安装

在以上条件准备好以后，就可以按照如下顺序进行开发环境的硬件连接了。

1、ES6210 采用 mini PCIe 全长卡尺寸，主板与底板的高度被控制在 5.5mm 之内，从而构成一套较紧凑的开发系统。

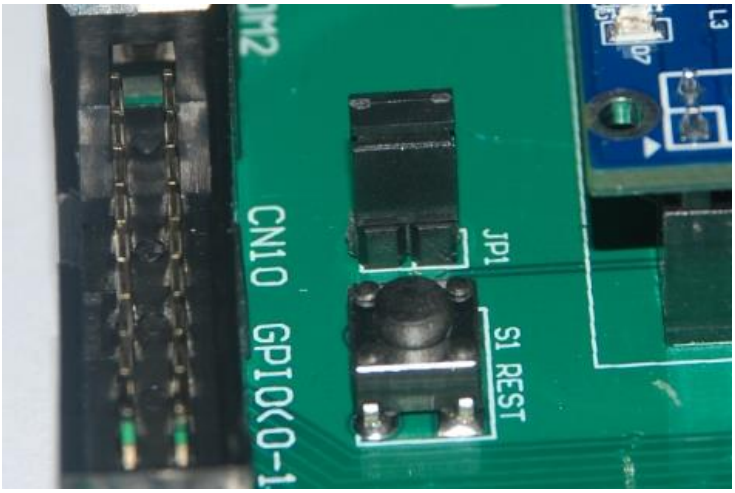
注：在用户收到的开发评估套件中，ES6210 往往已经插在开发评估底板上，开发过程中用户如需进行插拔，请注意插针和插座的序号对应。

2、ES6210 有两种工作模式：调试模式和运行模式。

调试模式是指开机以后系统处于调试状态，此时用户可以通过超级终端来操作 ES6210，实现应用程序下载调试、文件管理等功能。在开发阶段，系统总是处于这种状态。

运行模式是指开机以后系统自动开始执行用户指定的程序。开发完成，进入实际应用时系统总是处于这种状态。

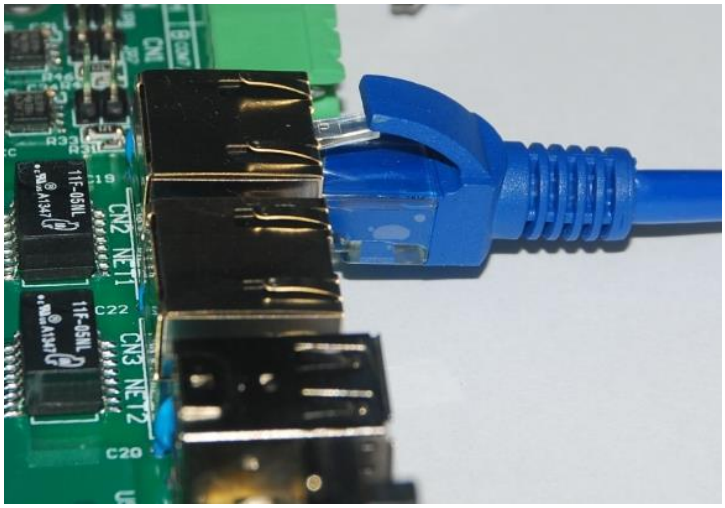
ES6210 工作于上述的哪一种模式，是通过开发评估底板上的跳线器 JP1 来选择的（JP1 在开发评估底板上的具体位置见下图）。JP1 短接，则工作于调试模式；JP1 断开，则工作于运行模式。



工作模式选择跳线器 JP1

3、用户可以用交换机/路由器/集线器将主机和 ES6210 接入同一个网络，如下图所示。这样开发主机和 ES6210 就能够建立起网络连接。

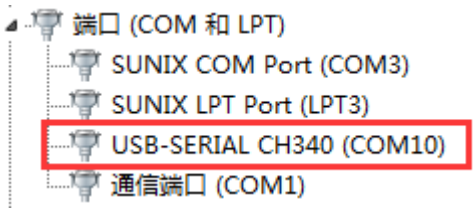
注：ES6210 的 ip 地址一定要与开发主机的 ip 地址设置在同一网段内。



将开发主机和 ES6210 接入以太网

至此，ES6210 运行的基本硬件环境已搭建完成。

现在可以给 ES6210 通电，即将 USB 线的 type Mini B 接口插在底板上的 Mini-B 接口里，此时，ES6210 上的红色电源 LED 指示灯亮，PC 上也会识别出相应端口，用于调试 ES6210。



PC 上识别 ES6210 的调试端口

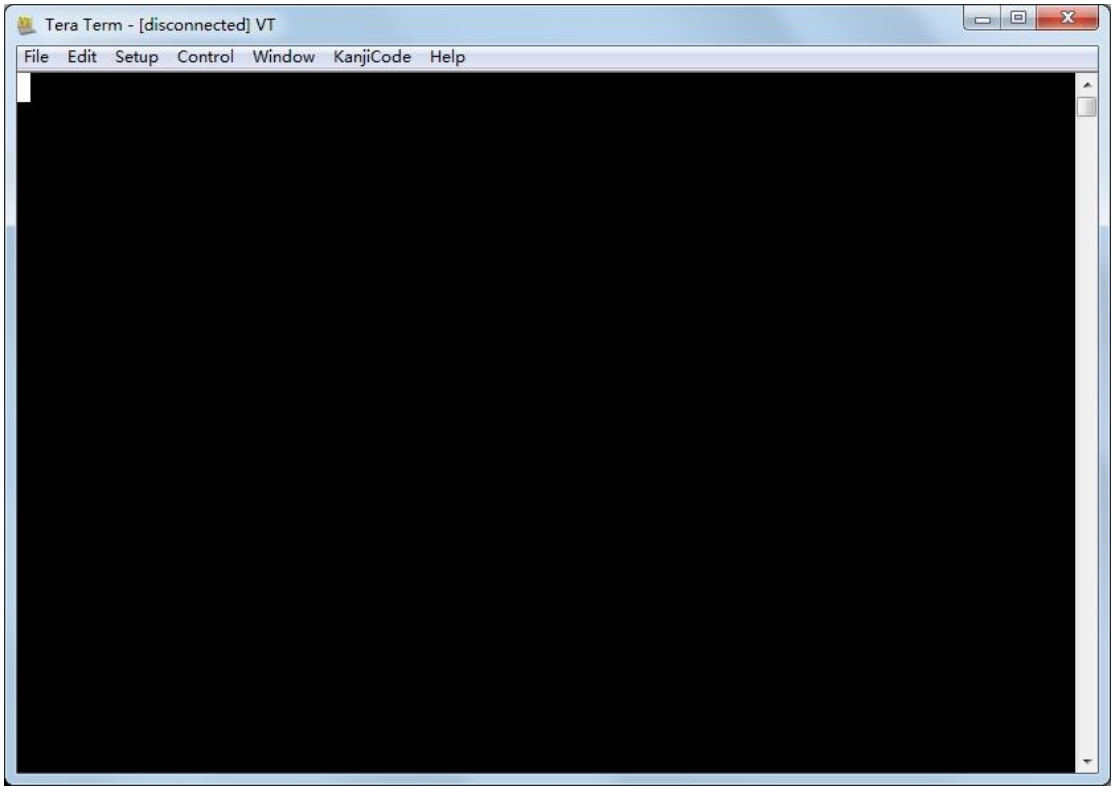
3 配置软件开发环境

ES6210 板载嵌入式 Linux-4.1.15 实时多任务操作系统，用户可以在主机使用 Windows 或者 Linux 操作系统进行应用程序的开发。鉴于 Windows 操作系统的广泛使用，为使用户快速、便捷地开发出自己的应用程序，减少学习 Linux 所需的时间和精力，英创公司进行了大量富有成效的工作，最终选取了一系列可以在 Windows 操作系统中开发 Linux 应用程序，并将程序下载到 ES6210 中运行测试的工具，下面将逐一介绍这些工具的安装、使用方法以及相关事宜，用户跟随本章的步骤即可快速搭建起 ES6210 的软件开发平台。

3.1 配置串口工具

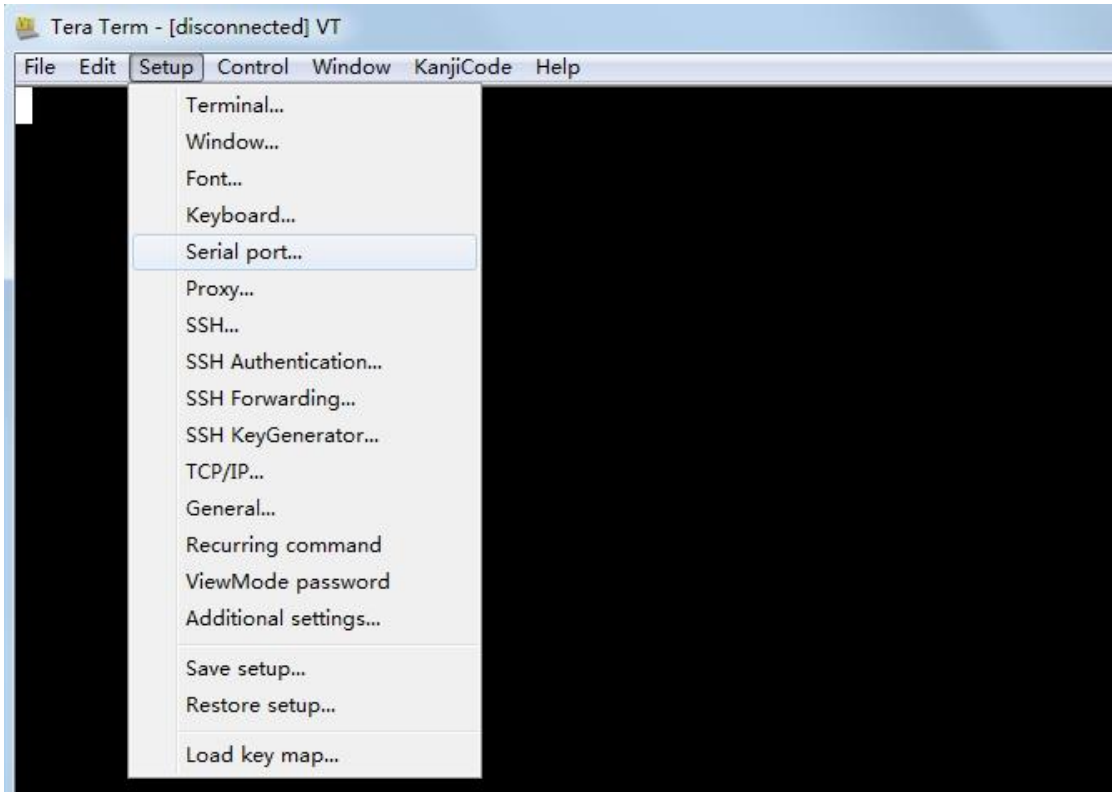
ES6210 的运行信息会通过串口工具显示在开发主机的显示屏上；用户想要对 ES6210 的文件系统进行操作也需通过超级终端以命令行方式进行。现在广泛使用的 Window 7 系统没有自带的串口工具，所以需要用户自己下载串口工具进行调试。英创公司推荐一款名为 Tera Term VT 的串口工具，并以 Tera Term VT 介绍配置的过程。

1、打开串口工具，如下图所示。

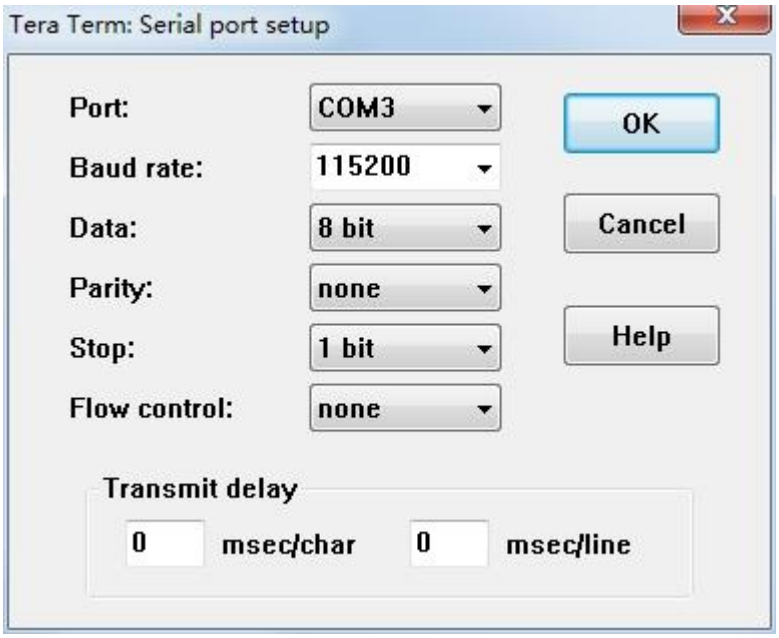


串口工具界面

2、如下图所示，选择 Step，点击子菜单 Serial port 新建串口。

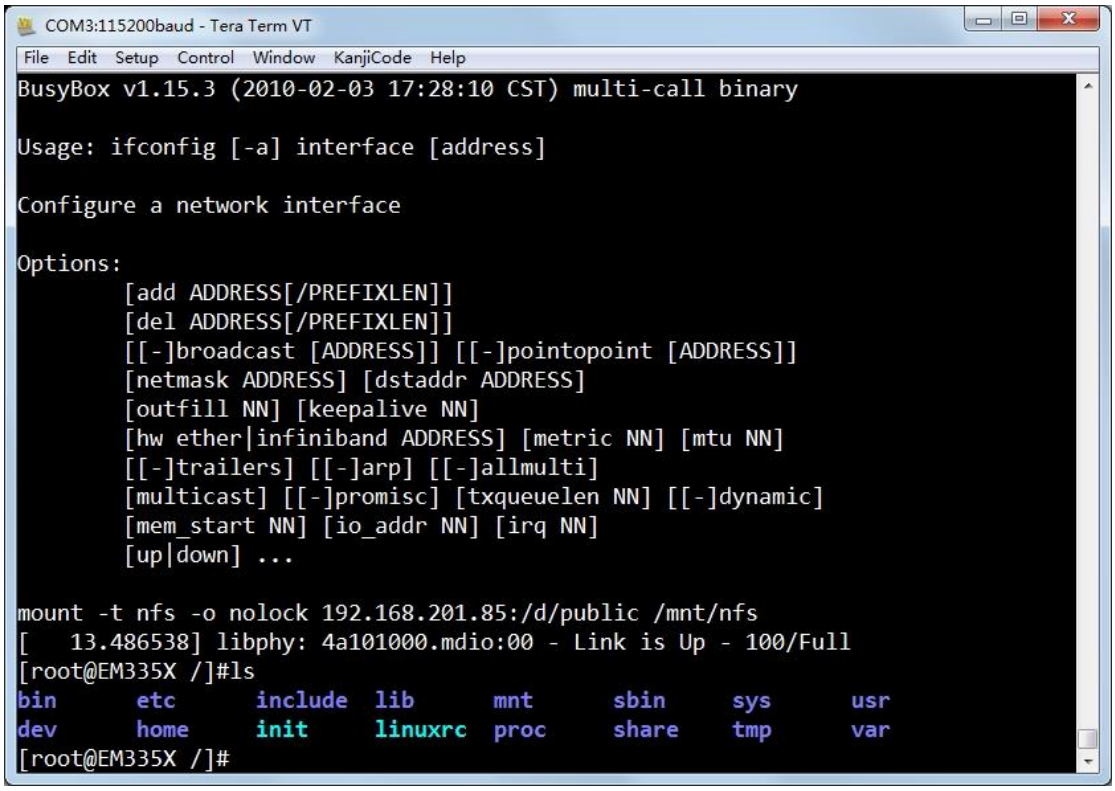


3、选择要使用的串口，这里使用的是 COM3，设置**每秒位数(Baud rate)**为 **115200**，**数据流控制(Flow control)**为**无**。



COM3 参数配置

4、完成以后给 ES6210 上电，超级终端将显示出 ES6210 的开机启动信息。启动成功以后回车进入命令行，此时可以通过超级终端使用 Linux 的命令对 ES6210 进行操作，如下图所示。图中显示的是使用 **ls** 命令查看 ES6210 中的文件。



通过超级终端使用 Linux 的命令操作 ES6210

3.2 编辑 userinfo.txt 文件

userinfo.txt 文件有三个作用：

1、配置 ES6210 的网络参数，让 ES6210 与开发主机处于同一网段

2、配置 NFS 挂载参数，让开发主机的指定目录能挂载到 ES6210 的指定目录下

3、配置应用程序参数，这样开发完成以后 ES6210 将自动根据该参数执行应用程序

userinfo.txt 文件的内容及格式如下（双斜线后不同字体和颜色的文字为加注的说明文字，并不包括在 userinfo.txt 文件中）：

```
[LOCAL_MACHINE]                    // ES6210 信息
DHCP="0"                           // 配置 DHCP 客户端信息。设为“0”则 DHCP 关
```

```
                                    // 闭，用户需手动设置网关、IP 地址、子网掩码；
                                    // 设为“1”则 DHCP 开启，ES6210 将自行获取
                                    // 上述网络参数
DefaultGateway="192.168.201.20"     // 默认网关，根据用户所在的实际运行网络设置
IPAddress="192.168.201.90"          // ES6210 的 IP 地址，由用户自行设置
SubnetMask="255.255.255.0"          // 子网掩码，根据用户所在的实际运行网络填写
[NFS_SERVER]                        // NFS 挂载信息
IPAddress="192.168.201.85"          // 开发主机 IP 地址，根据用户所在的实际运行
                                    // 网络设置
Mountpath="/d/public"               // 开发主机上被挂载的文件夹名，本文中以
                                    // “public”为例，用户可自行选择任意文件夹，
                                    // 需注意的是必须带上文件夹路径
[USER_EXE]                          // 用户程序信息
Name="/mnt/nandflash/hello"         // 系统开机自动执行的程序及其存储路径。开发
                                    // 完成以后用户将自己的应用程序文件名填在
                                    // 双引号之间取代目前的默认文件名，开机即可
                                    // 自动运行（注意，用户也可以在
                                    // /mnt/nandflash/下建立子目录存放应用程序，
                                    // 配置此项参数的时候一定要带上绝对路径）
Parameters=""                       // 系统开机自动执行的程序的参数配置。开发完
                                    // 成以后在此处填入实际应用程序的参数，如果
                                    // 没有则不填，但必须保留双引号
```

根据用户的实际网络参数编辑好 userinfo.txt，存入 U 盘，将 U 盘接入 ES6210 开发评估底板的 USB 接口，短接 JP1 使 ES6210 处于调试模式，然后上电。系统将自动搜索 USB 接口，将读到的 userinfo.txt 文件存放到/mnt/nandflash 目录中，并按照其内容配置 ES6210 的网络参数。启动完成以后，可以通过超级终端使用 **ifconfig** 命令查看是否配置完成。

userinfo.txt 写入 ES6210 以后，系统每次开机都会自动读取该文件并按照文件内容进行配置。如果其中任何参数需要重新配置，可编辑好 userinfo.txt 并重复执行上述步骤。

如果要让系统开机自动挂载，则 ES6210 上电启动之前必须先打开 WinNFSd.exe。

如果 ES6210 处于运行模式，则开机以后会自动执行 **Name="/mnt/nandflash/"**中设置的应用程序。英创公司为用户分配的存储地址固定在**/mnt/nandflash** 文件夹下，用户可以将应用程序直接存在这个目录中，也可以在此目录下建立子目录存放应用程序。用户配置该项参数的时候要带上绝对路径，否则系统无法找到执行文件。

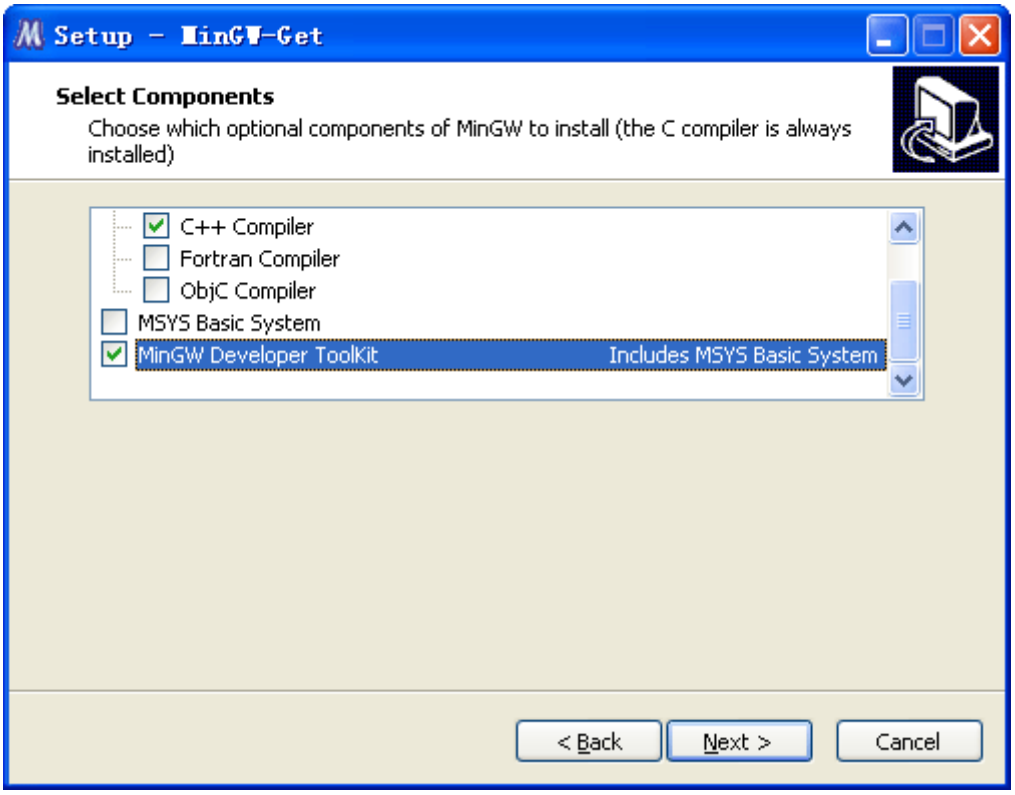
注：Linux 操作系统严格区分大小写，因此此处的用户应用程序名称必须与实际的程序名称完全一样，包括大小写字母。

3.3 安装 eclipse

eclipse 是一款开源的免费开发工具，可以直接在 Windows 操作系统下生成 Linux 应用程序，省却用户学习使用 Linux 开发工具所需的大量精力和时间。经英创公司的努力工作，eclipse 编译的程序已经可以在英创主板上直接运行。此外，eclipse 的开发环境和 Visual Studio 等常用开发工具很相似，用户可以通过市面上很常见的 C 语言书籍以及 Linux 书籍来进行编程方面的学习。

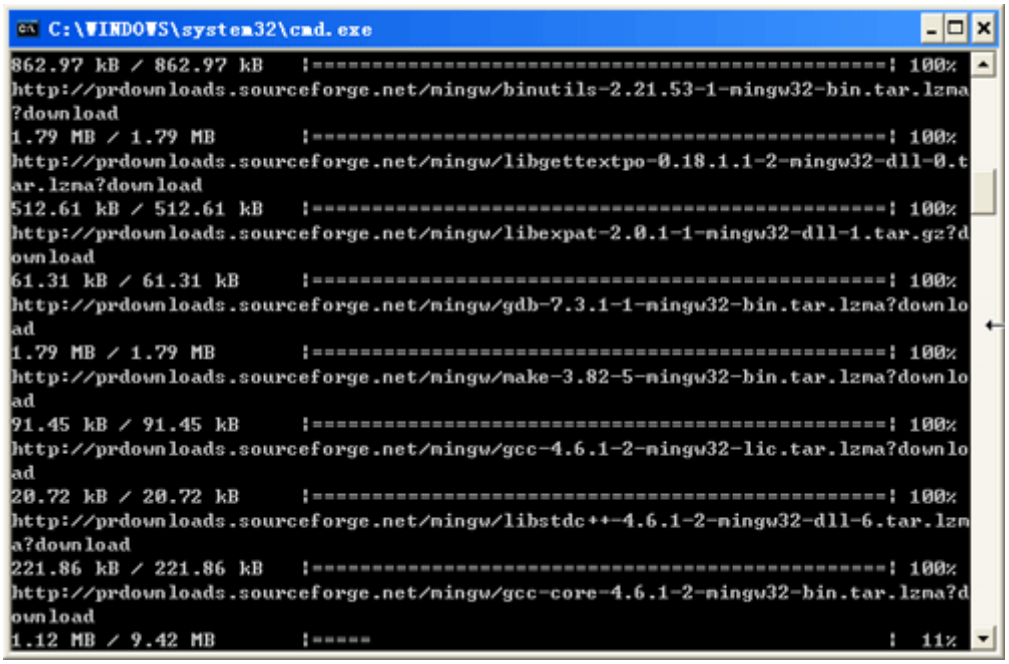
在常规的 Linux 应用程序开发中，一般是以命令行方式，首先编写 Makefile 文件，然后通过 make 工具来运行 GCC，完成对程序的编译链接的。这种编程方法由于门槛较高，使 Linux 的应用范围受到相当的限制，特别是对技术力量相对薄弱的中小企业更是如此。而当采用 eclipse 时，用户已不再需要涉及复杂的 Makefile 文件的编写，可把精力集中在应用程序本身实现的功能上，从而大大加快了应用程序的开发进度。

1、在开发光盘的“工具软件”中找到“EclipseOnWindows”文件夹，安装 mingw-get-inst-20111118。如下图所示，在 **Select Components** 对话框中选择 **C Compiler**（该项默认已选中）、**C++ Compiler** 和 **MinGW Developer ToolKit** 三项，其他所有配置均使用系统默认设置。



安装 MinGW

MinGW 安装完成以后，会自动从网络下载一些库文件，如下图所示。等下载完毕以后，再继续后续软件的安装。



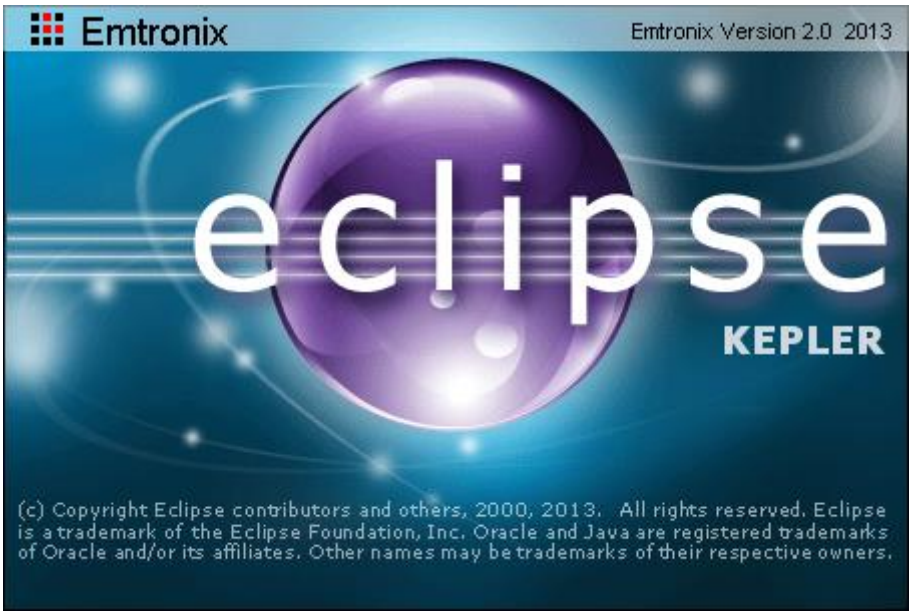
MinGW 自动下载文件

2、MinGW 安装完成以后，安装 jre-7u2-windows-i586，所有配置均使用系统默认设置。

3、jre 安装完成以后，安装 arm-2009q3-67-arm-none-linux-gnueabi，所有设置全部采用系统默认设置。

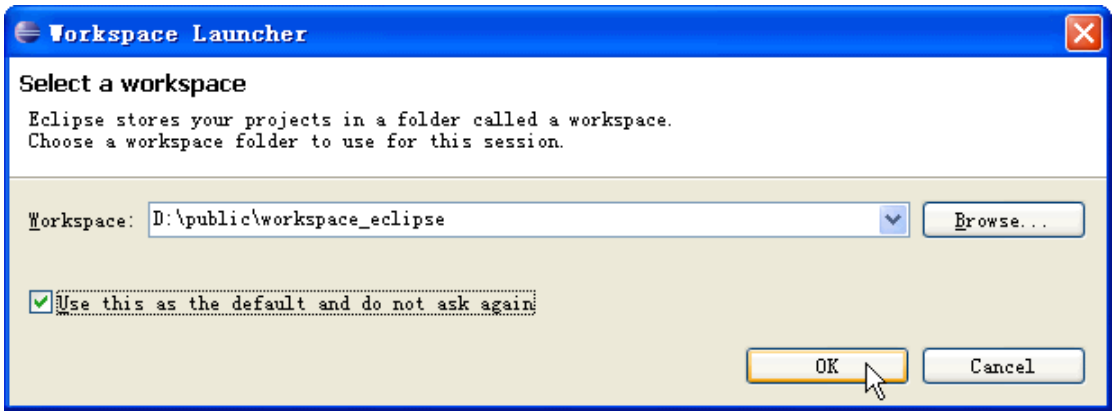
4、G++安装完成以后，将“eclipse”文件夹复制到开发主机任意目录（路径中不能有中文，建议放到 C 盘根目录下）。

5、完成以后进入 eclipse 文件夹，启动软件，用户可以看到加载了英创公司信息的启动画面，如下图所示。



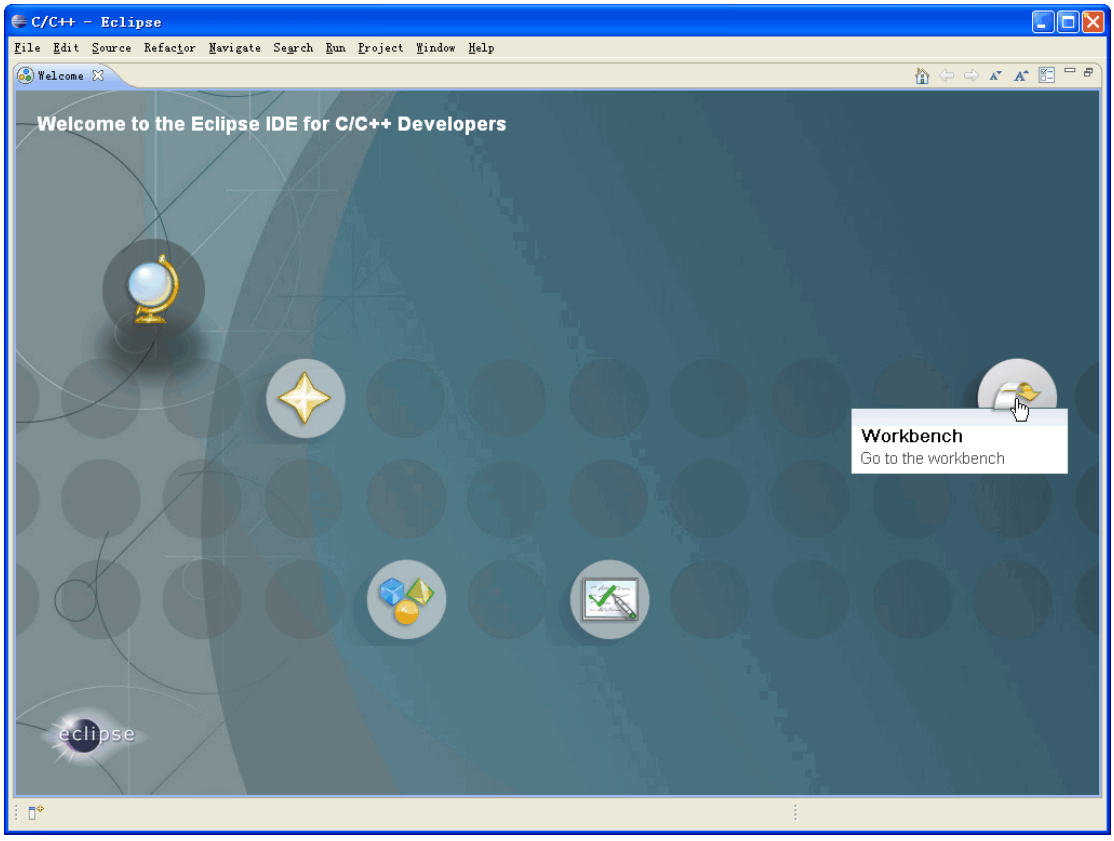
加载了英创公司信息的 eclipse 启动画面

6、出现如下图所示对话框，指定工程文件的默认保存路径，此后所有使用eclipse设计的文件将自动保存到该文件夹下。本文中以**D:\public\workspace_eclipse**为例，用户可自行指定，但是注意路径中不要带有中文名。



选择工程文件保存路径

7、启动以后，进入 Workbench，如下图所示。



进入 Workbench

至此，eclipse 开发工具安装完成。

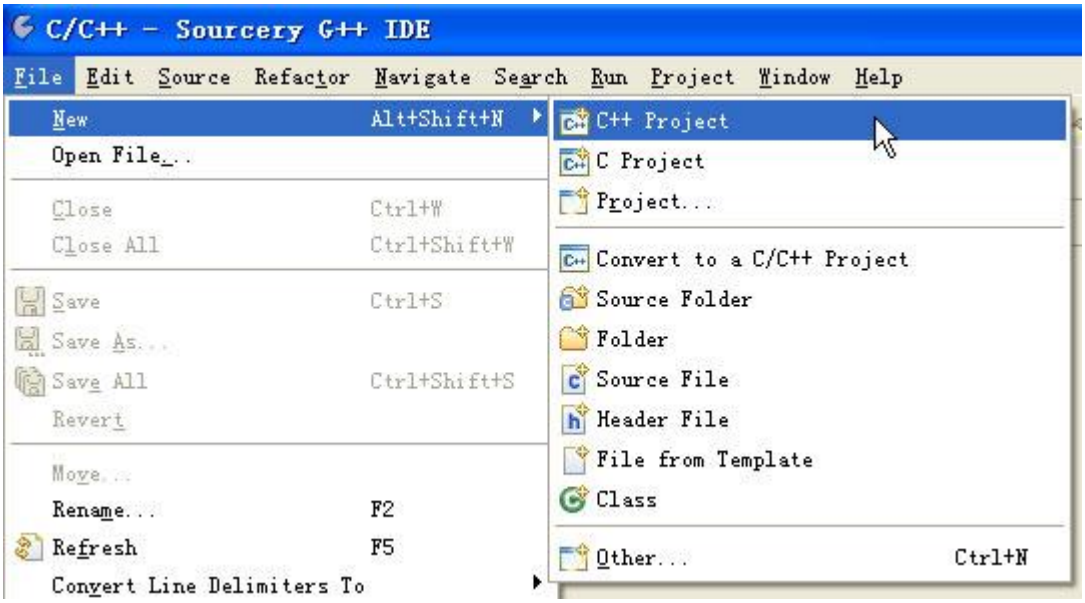
3.4 eclipse 下创建和管理 C++应用工程

在进行应用程序开发时，往往会根据不同的功能把应用程序划分成多个模块来进行设计，而每个模块通常对应一个源文件，这样有利于应用程序的管理和维护，对于具有多个模块的应用程序的管理一般是通过应用工程文件进行的，因此每一个应用程序对应着一个应用工程文件。本章主要是通过介绍创建、修改、编译应用程序工程文件的方法，对 eclipse 的使用进行详细说明。

对一个全新的 Linux 应用程序开发，用户可创建全新的工程文件，也可利用已有的工程文件进行必要的修改，来完成应用程序的开发。

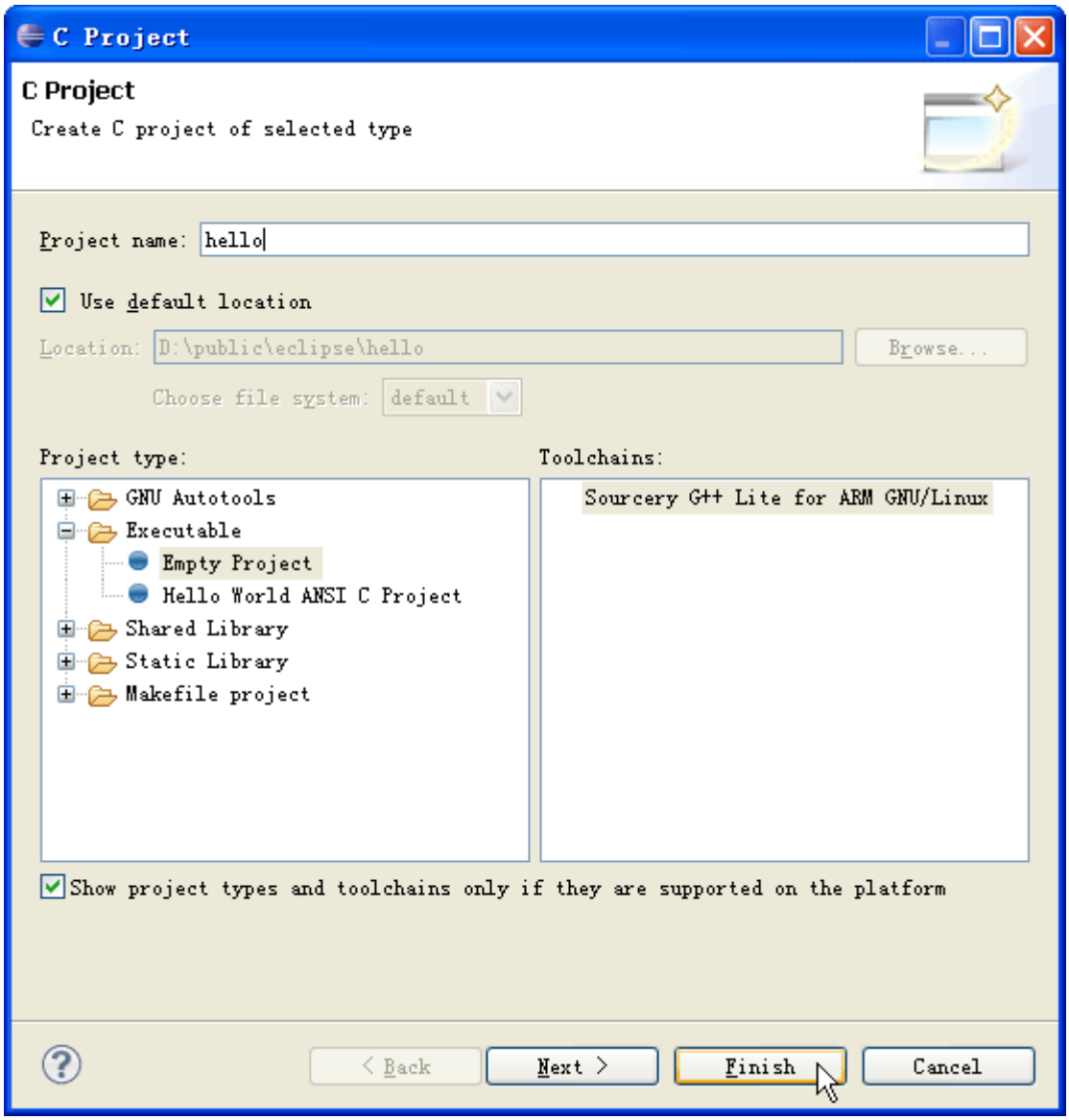
创建一个新的工程文件

1、创建一个新的工程文件，是通过选择 **File -> New -> Project…**来实现的，我们建议客户选择 **C++ Project**，如下图所示。



建立新的 C++工程文件

2、接下来输入工程文件的名称以及相应的位置路径，在这里建议不要选用系统的缺省路径，而是选择用户自行建立的 Linux 应用程序开发路径，以便于应用程序的管理。然后选择 **Project type: -> Executable -> Empty Project**。在建立工程的时候，在 **Toolchains:** 中一定要选择 **Sourcery G++ Lite for ARM GNU/Linux**，如下图所示。



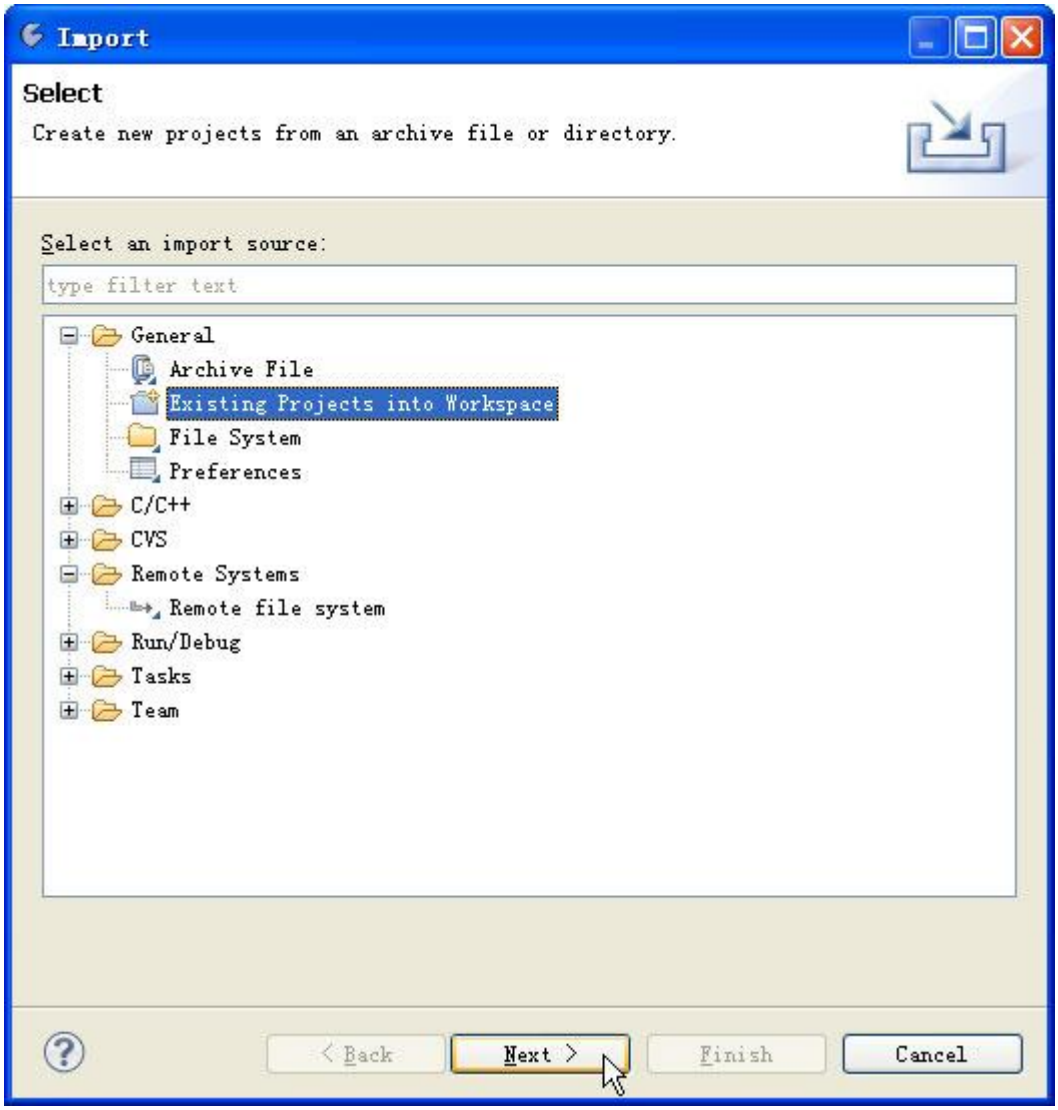
选择 Toolchains

3、点击 **Next** 进入 G++属性设置菜单，选择目标机的处理器类型、大端或小端模式以及浮点处理支持等。对于 ES6210 来说，这些设置均可以采用缺省配置。

4、然后直接点击 **Finish**，一个新的工程文件就建立完毕了。

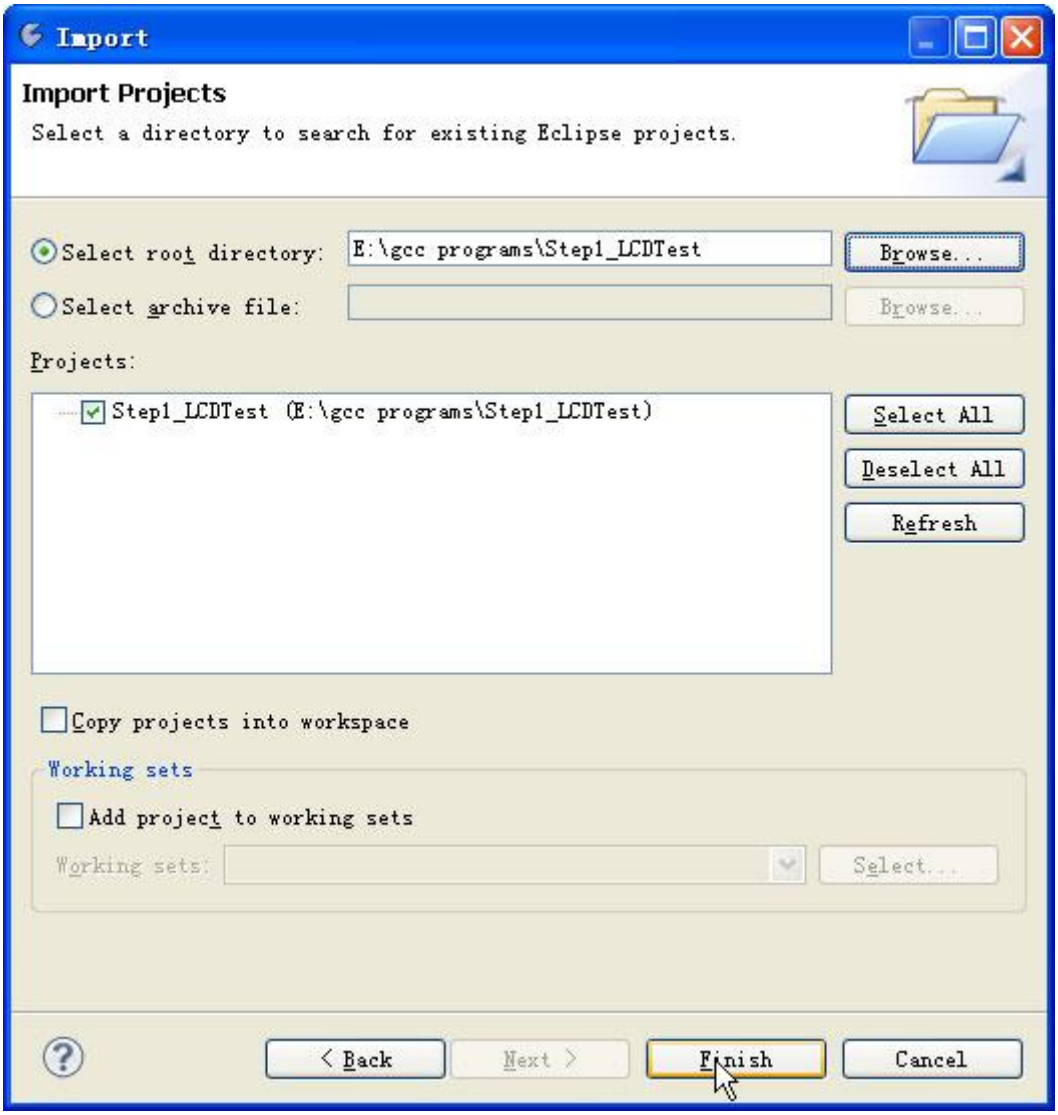
打开已有的工程文件

1、选择 **File -> Import…**，在 **General** 下选择 **Existing Projects into Workspace**，如下图所示。



打开已有的工程文件

2、点击 **Next**，通过 **Browse…**选择已有的工程文件的目录，然后点击 **Finish** 打开图中显示的 Step1_LCDTest 应用工程，如下图所示。



找到工程文件并打开

添加新的源文件

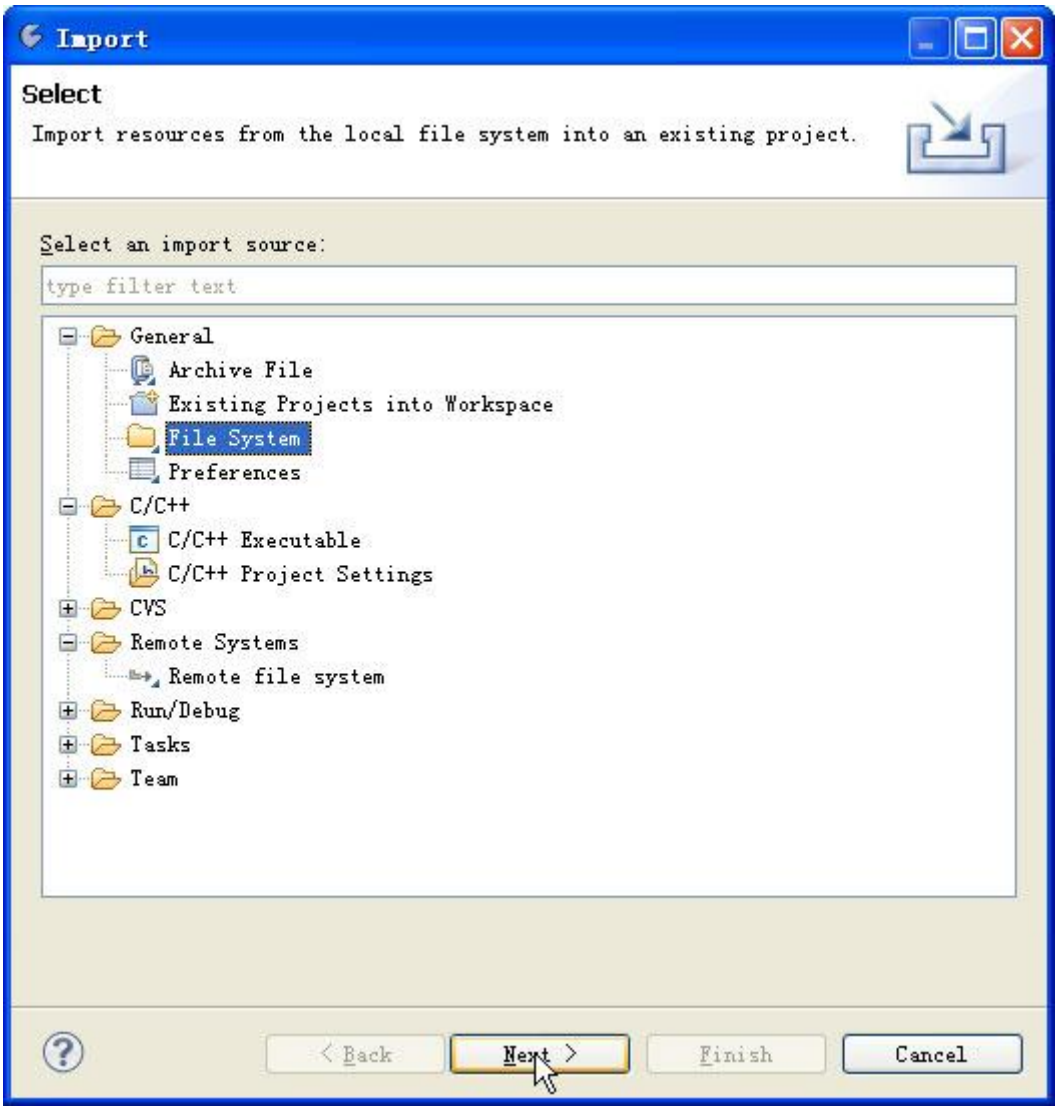
在 Project Explorer 视窗下选择需要添加源文件的工程，然后点击鼠标右键，选择 **New -> Source File** 或者 **New -> Header File**，即可增加新的 CPP 或者 H 文件到该工程中。

编辑完成 CPP 文件或者 H 文件后，可以通过菜单中 **File -> Save** 或者通过按 **Ctrl+S** 键进行保存。

添加已有源文件

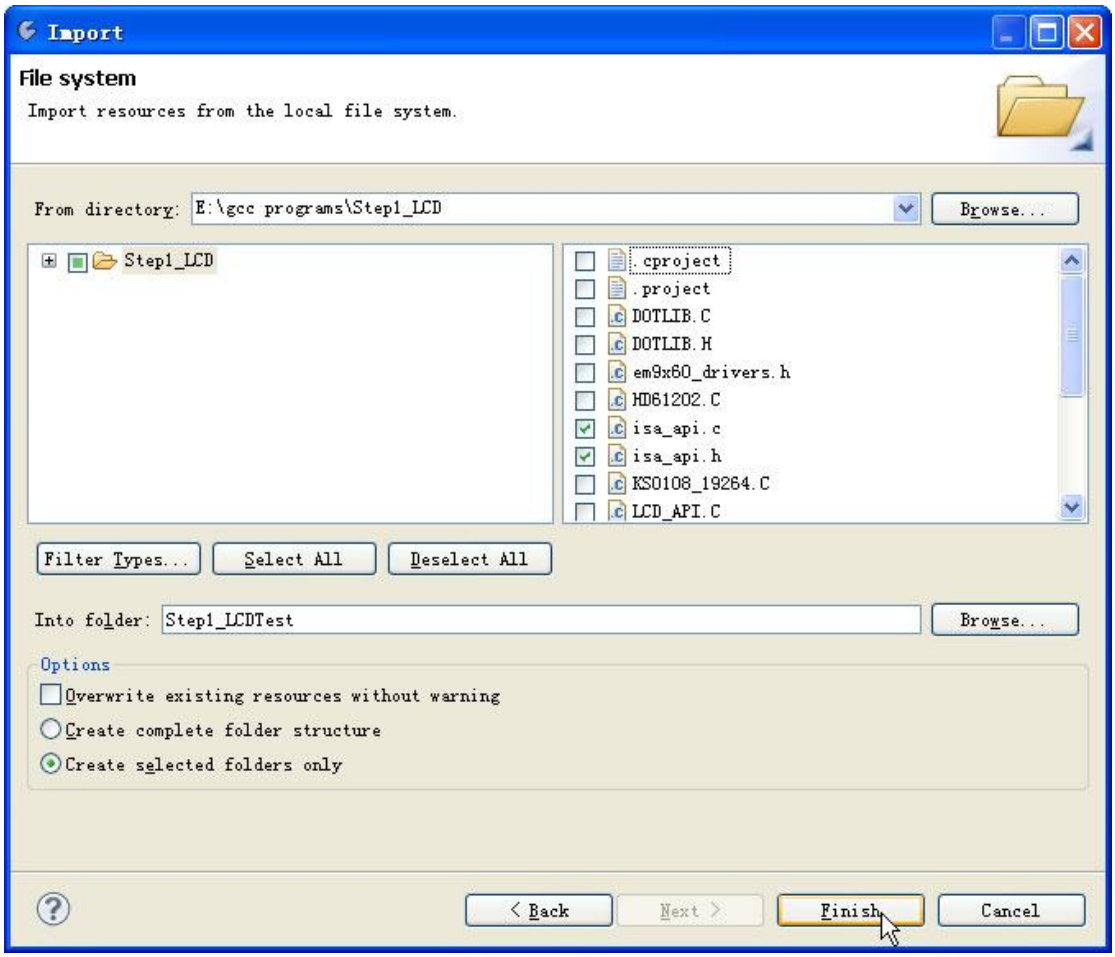
1、在 Project Explorer 视窗下，选择需要添加已有源文件的工程，然后点击鼠标右键，

选择 **Import…**，选择 **General -> File System**，如下图所示。



添加已有的源文件

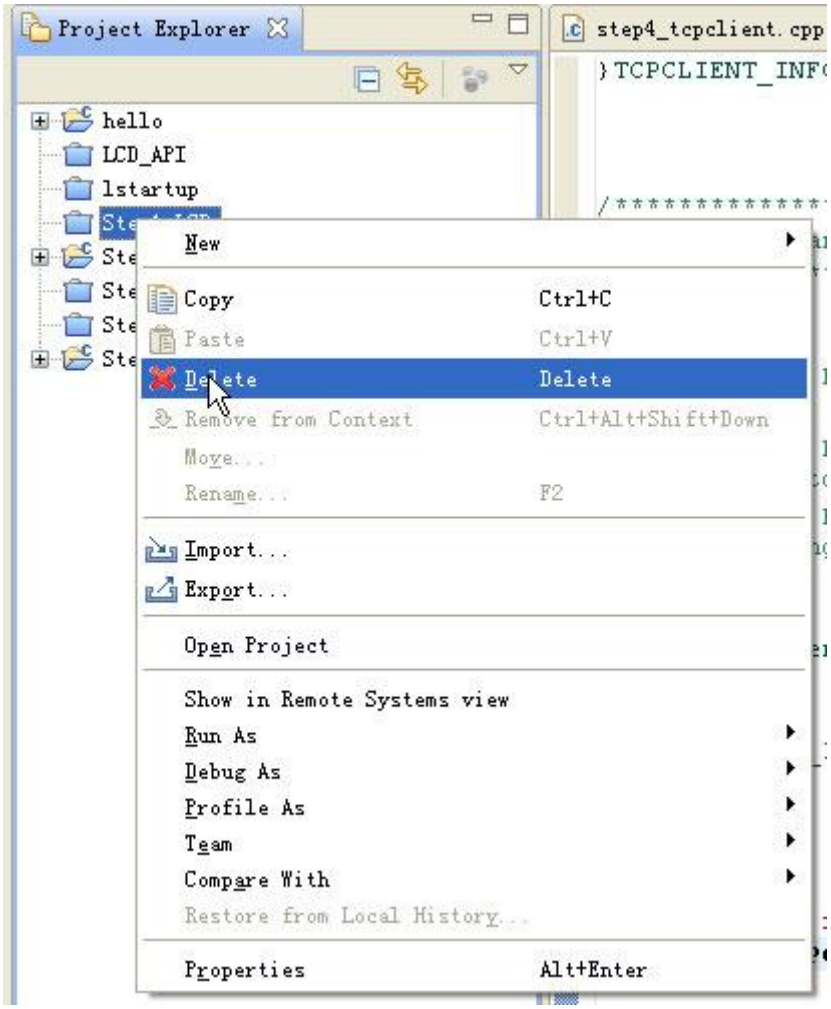
2、通过打勾进行文件的选择，如下图所示。



选择要添加的源文件

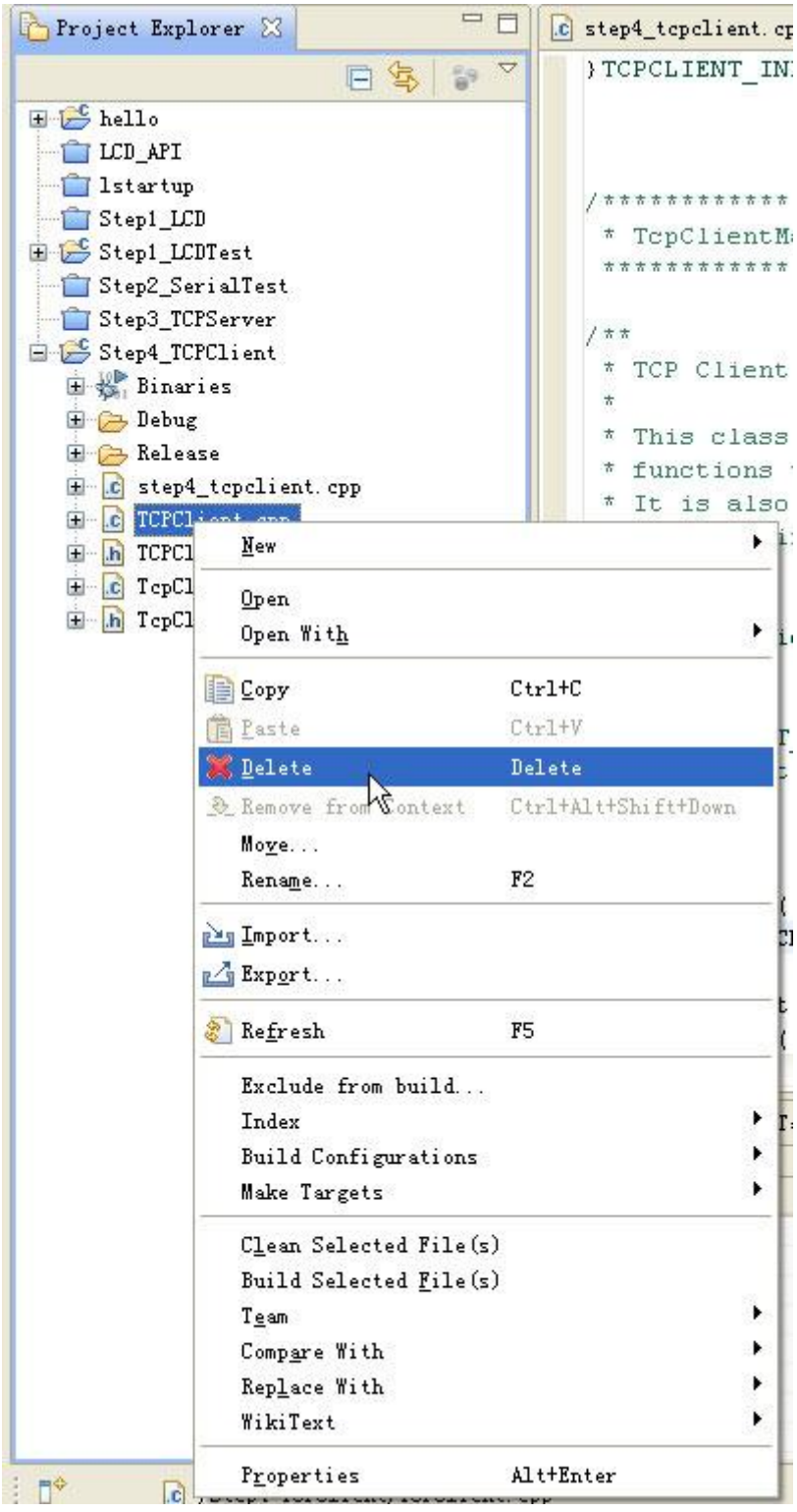
删除文件

删除工程文件：在该工程文件上点鼠标点键，然后选择 **Delete**，如下图所示。



删除工程文件

删除工程文件中的源文件：将鼠标移至该文件处，然后点击鼠标右键，选择 **Delete**，如下图所示。

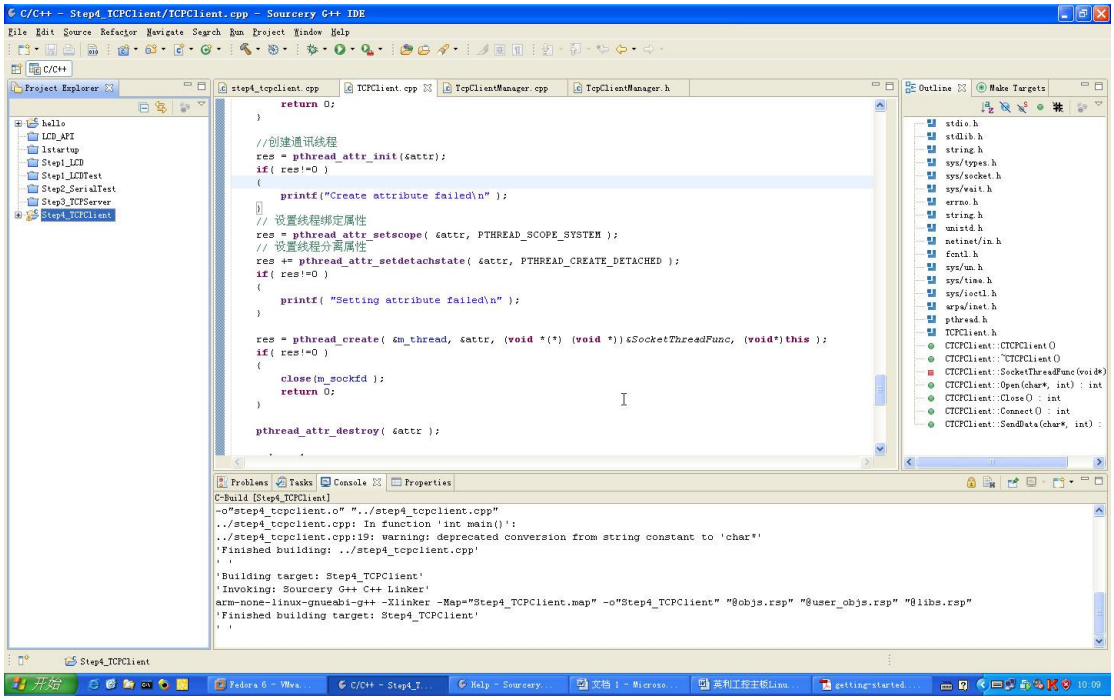


删除源文件

3.5 eclipse 下编译 C++应用工程

工程文件编辑完成后，就在 eclipse IDE 环境下直接进行编译生成相应的可执行文件。

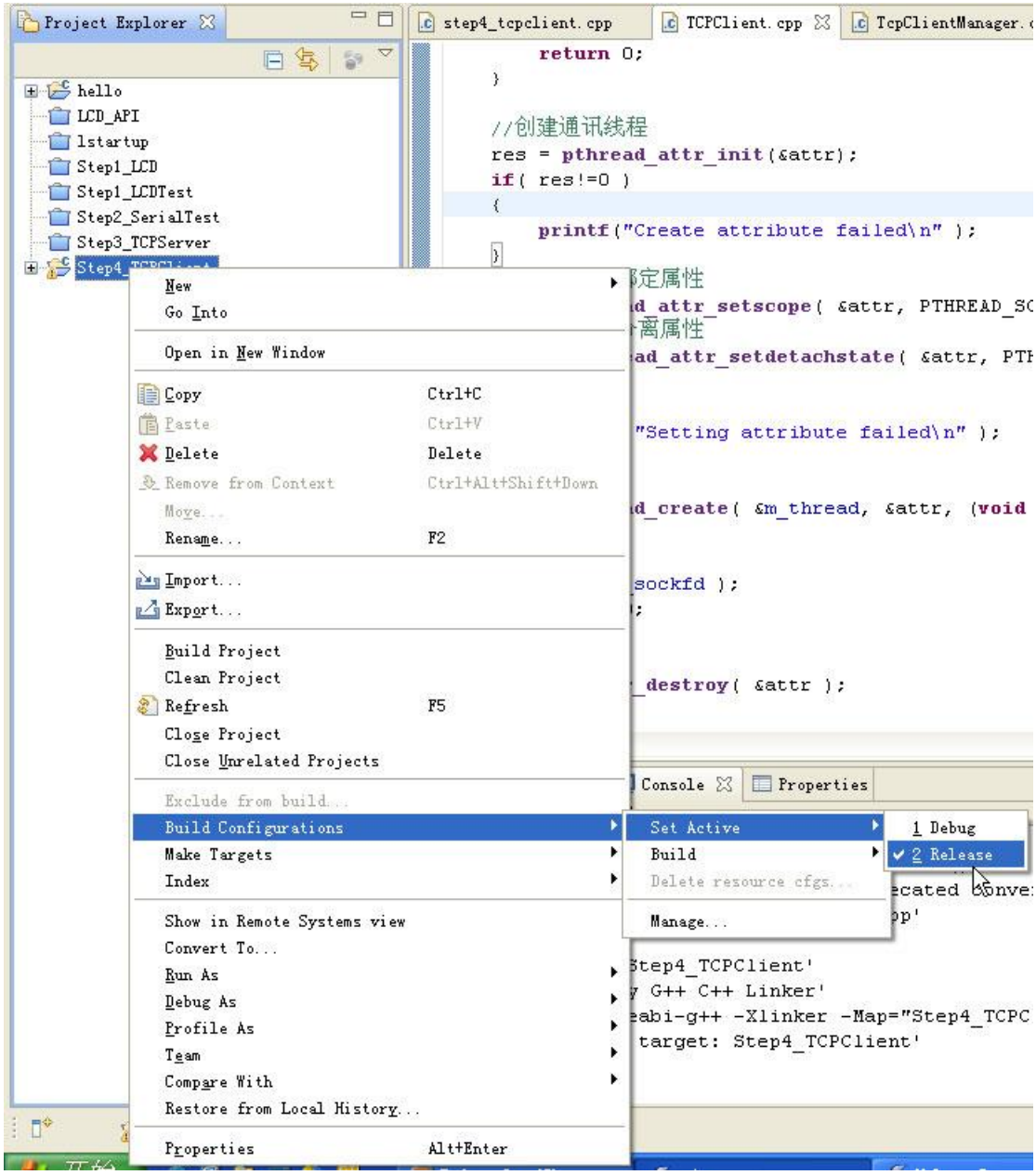
在 Project Explorer 窗口下选择需要编译的工程文件，如下例选择 Step4_TCPClient 工程，然后点击菜单中 **Project -> Build Project** 即可进行程序的编译，如下图所示。也可以选择使用工具栏中的  图标。编译输出的结果显示在 Console 窗口中，如果因为某种原因该窗口没有显示，可以通过选择 **Window -> Show View -> Console** 来实现。



编译工程文件

编译模式的选择

选择应用工程编译的 Debug（调试版本）和 Release（发行版本）。在 Project Explorer 视窗下，选择需要调试的工程文件，然后点击鼠标右键，选择 **Build Configurations -> Set Active -> Release** 项，如下图所示。

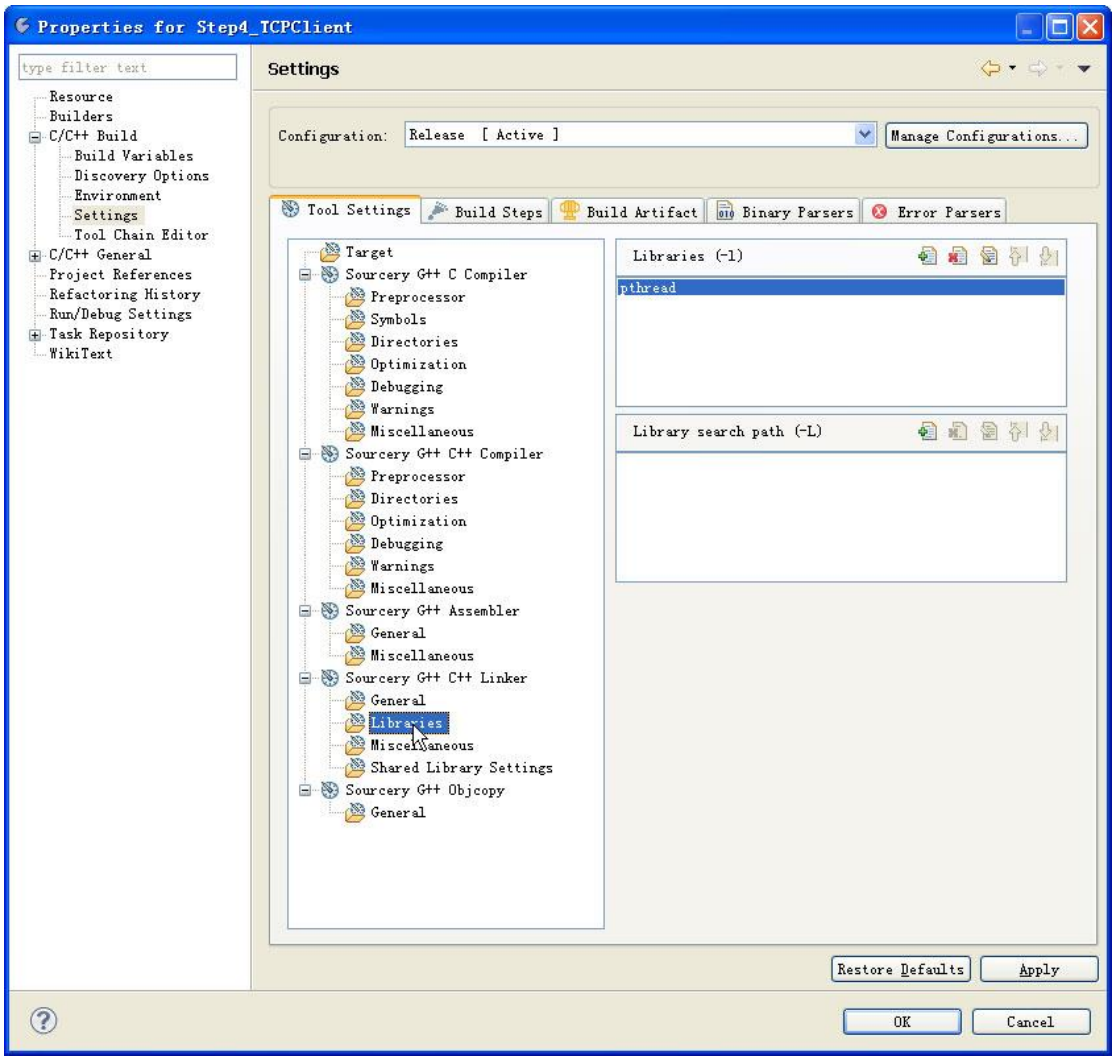


编译模式选择

编译属性的设置

设置应用工程编译属性，在实际应用中用得比较多的是应用工程需要 Link 专用的库文件，此时就需要对 C/C++的编译属性进行相应的设置。

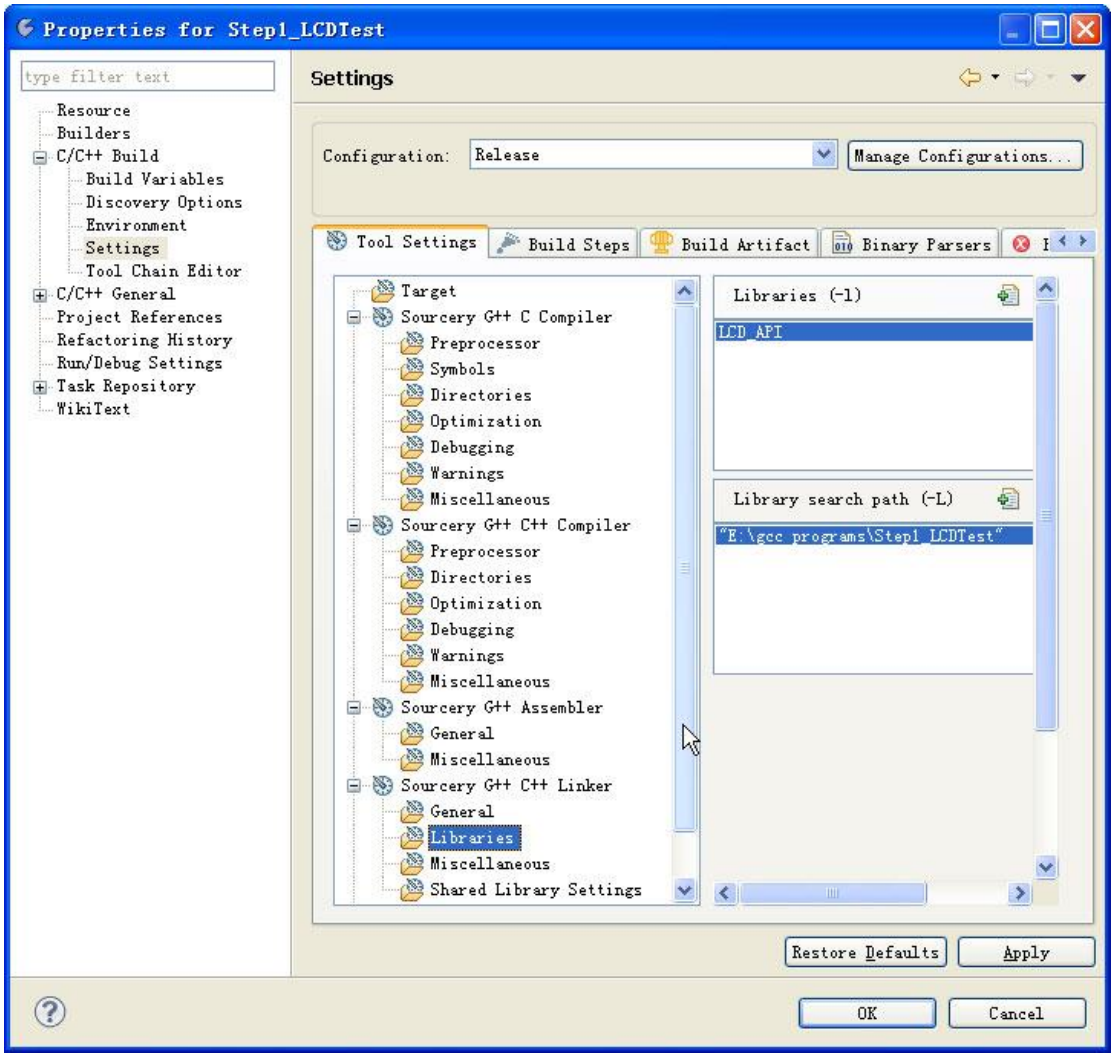
在 Project Explorer 视窗下，选择需要设置的工程文件，然后点击鼠标右键，选择 **Properties** 项，在窗口中选择 **C/C++ Build -> Settings -> Tool Settings -> Sourcery G++ C++ Linker -> Libraries**，如下图所示。其中的一个窗口用于指定库文件的名称，一个用于指定库文件的路径，对于系统中已有的库文件，就不需要指定路径；而对于用户专用的库文件，则需要指定路径。



设置库连接

在进行 Linux 应用程序开发时，如果采用了多线程的操作，进行编译前，需要把线程库 libpthread 添加到链接选项中。注意，在 G++集成环境的缺省设置中没有链接此库。

如果在应用程序中需要用到专用的静态库，也需要在此选项中进行设置。举例说明：如程序中需要用到 API 函数库 libLCD_API.a，该库文件放置在相应的应用工程目录下，此时编译属性中还需要设置 libLCD_API.a 库文件所在的文件路径，其设置如下图所示。



库文件属性设置

3.6 设置文件系统挂载

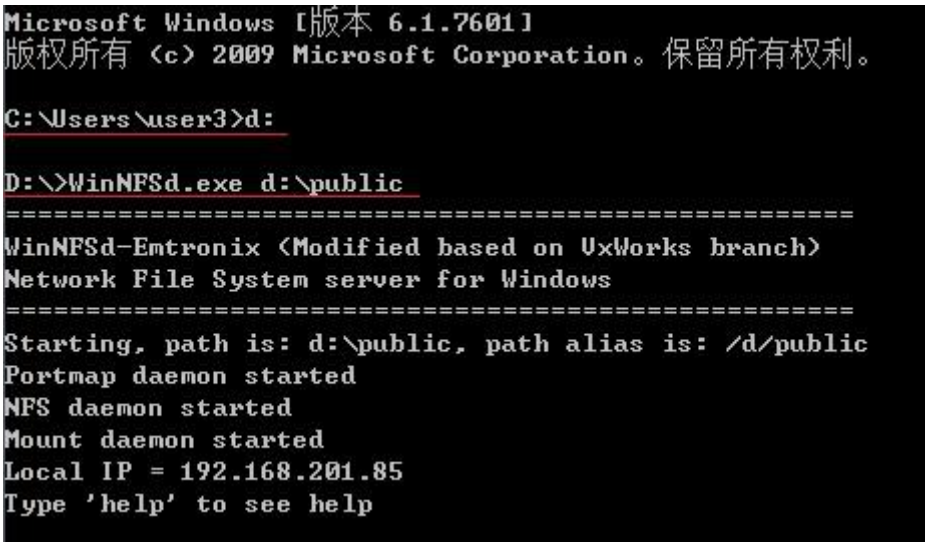
用户在开发主机中完成的应用程序必须通过一定的方法下载到 ES6210 的存储器中，才能进行运行测试。这种文件复制的方法有很多，英创公司建议使用文件系统挂载，此方法可以将开发主机中用户指定的某一个目录挂载到 ES6210 的 Linux 目录中，这样，用户在开发主机中完成的应用程序就可以直接放在该目录下，然后通过超级终端让其在 ES6210 上进行运行测试。

英创公司为用户提供的 NFS 服务器是一款名为 WinNFSd.exe 的免费 NFS 服务器，只需使用命令提示符打开和启动该服务器，即可以使用 NFS 功能。

1、打开开发资料光盘上的“工具软件”文件夹，将“WinNFSd.exe”复制到任意路径（路径不能带有中文名，本文以复制到 D 盘根目录为例）。

2、打开命令提示符，进入 D 盘，启动 WinNFSd.exe，如下图所示（此处的挂载文件夹以 d:\public 为例，用户可自行设置）。

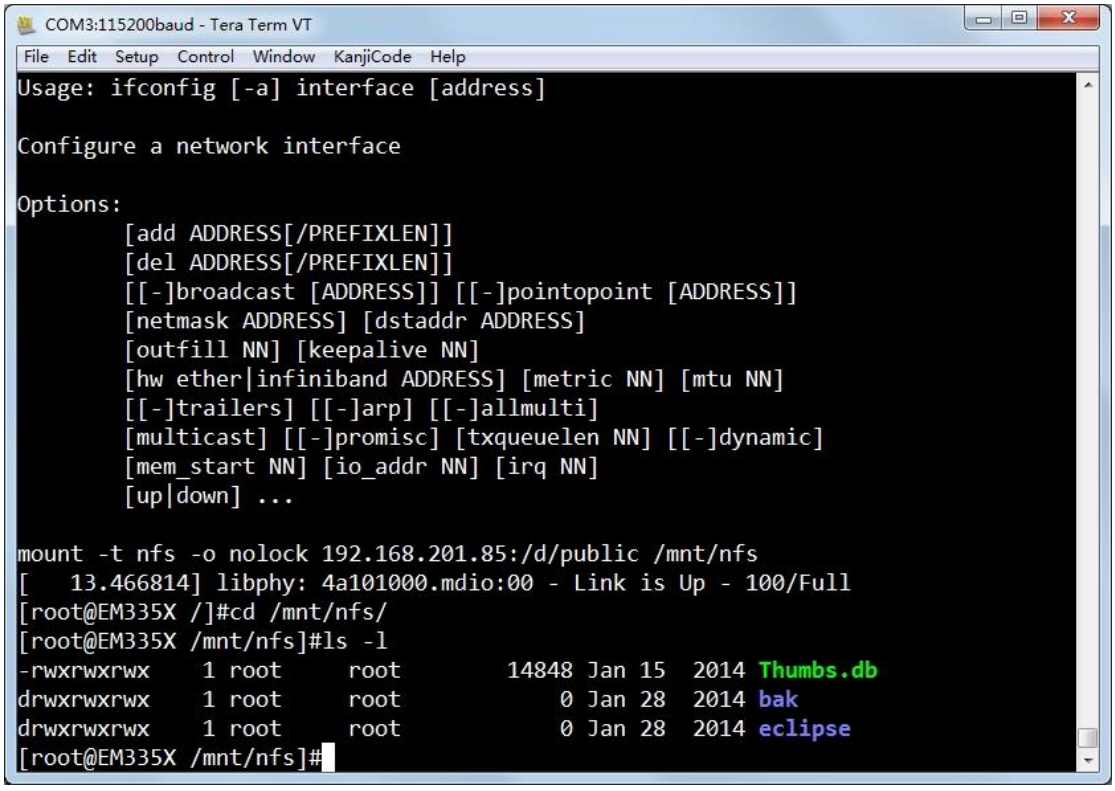
挂载文件夹一定要与 userinfo.txt 中 NFS_SERVER 项参数“Mountpath”中的配置一致。关于 userinfo 的编辑方法参见 2.2 节。



启动 WinNFSd.exe

3、确认 userinfo.txt 文件已配置好并存入 U 盘，将 U 盘接在工控主板的 USB 接口上，然后为系统上电。英创公司在 ES6210 上为开发主机指定的挂载点是**/mnt/nfs**，因此，在超级终端中使用命令 **cd /mnt/nfs** 进入 nfs 文件夹，使用命令 **ls -l** 查看，可以看到开发主机上 public 文件夹下的内容，如下图所示，表示挂载成功。

注：ES6210 上电启动之前必须先启动 WinNFSd.exe，才能自动挂载。



查看挂载到 Linux 目录下开发主机中的文件夹

4、如果开机挂载没有成功或者使用中连接中断，建议重启 WinNFSd.exe，然后键入命令进行挂载：

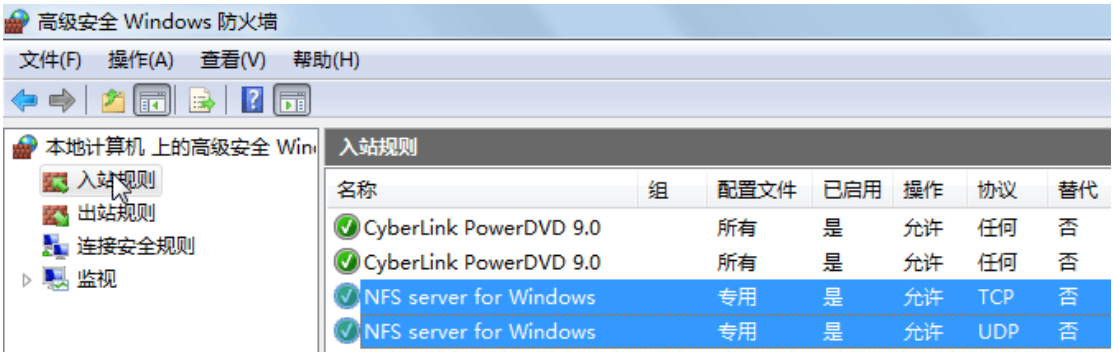
mount –t nfs –o nolock 192.168.201.85:/d/public /mnt/nfs

上述命令中的红字部分仅为示例，用户应填写自己实际的开发主机 IP 地址和挂载文件夹目录。

5、如果挂载仍然失败，建议同时重启工控主板和 WinNFSd.exe，然后再次测试。需注意的是，应确认该服务器没有被杀毒软件或者 Windows 自带的防火墙阻止。

Windows XP 防火墙中显示如下：

Windows 7 中防火墙显示如下：

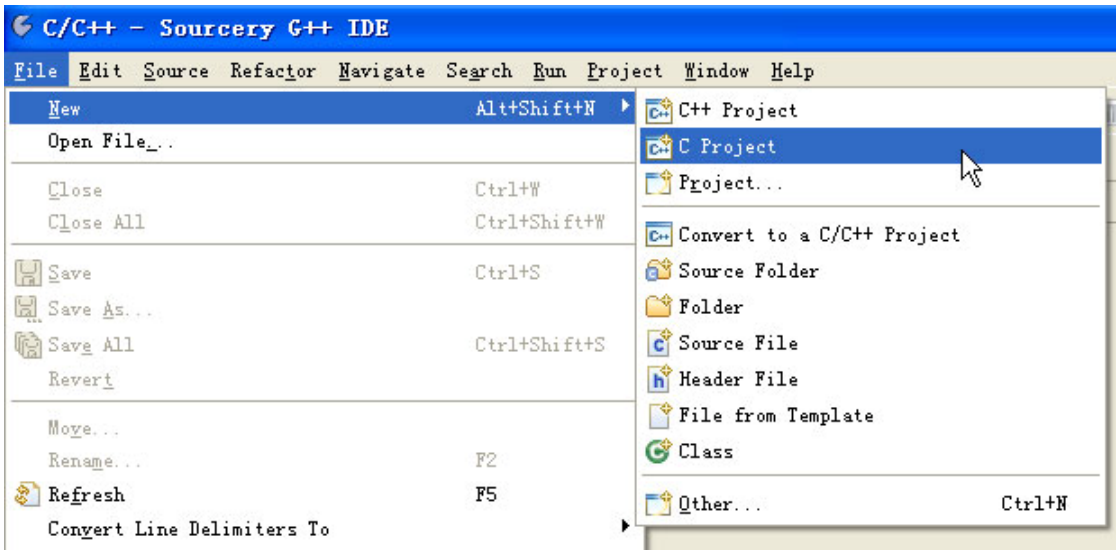


4 开发自己的应用程序

经过前两章，ES6210 的软硬件开发环境搭建均已完成，接下来用户可以进行应用程序的开发了。本章将通过两个实例介绍 ES6210 的软件应用开发步骤。

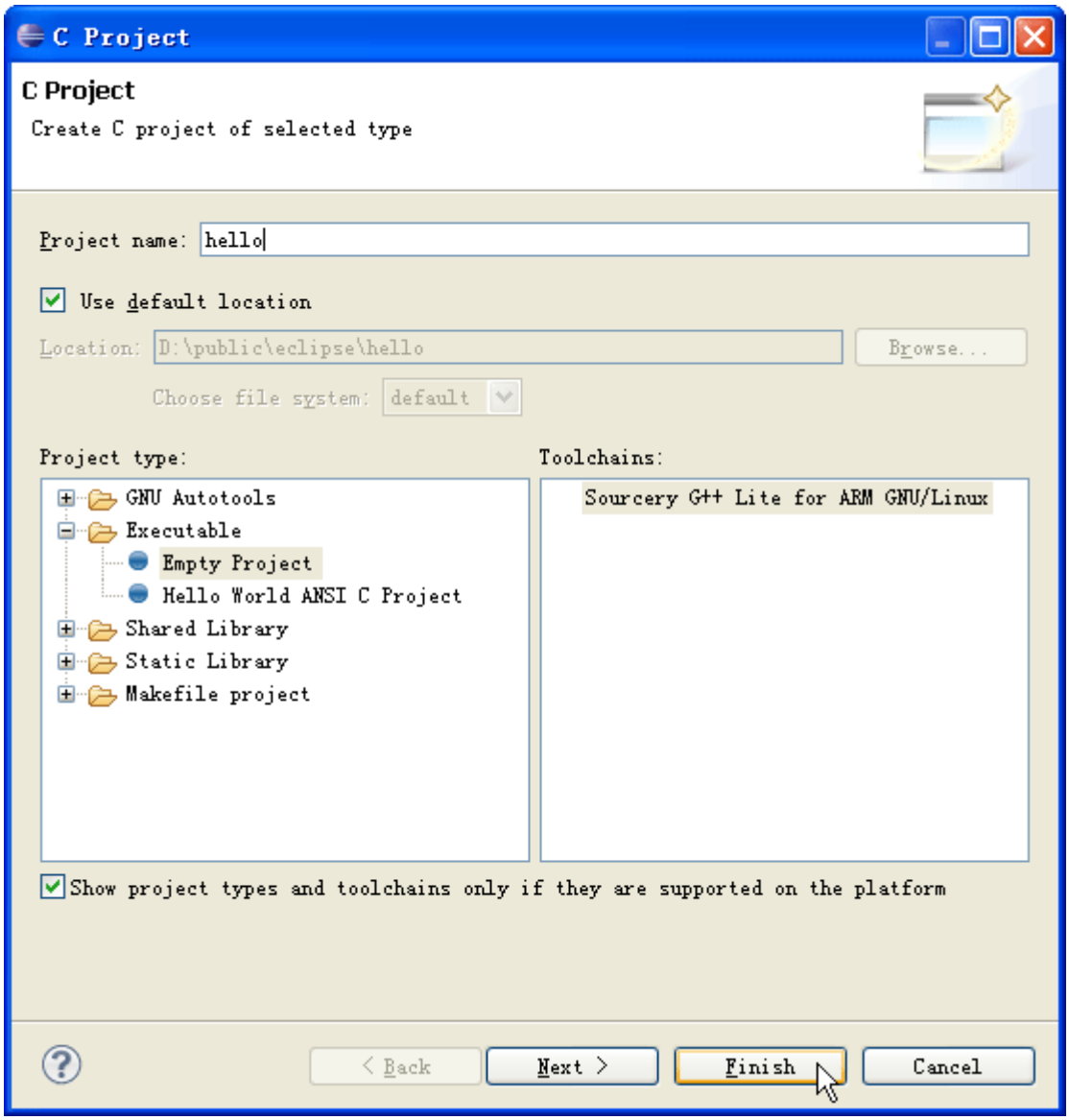
4.1 创建工程文件 hello

1、启动 eclipse / Sourcery G++，选择 **File** -> **New** -> **C Project**，如下图所示。



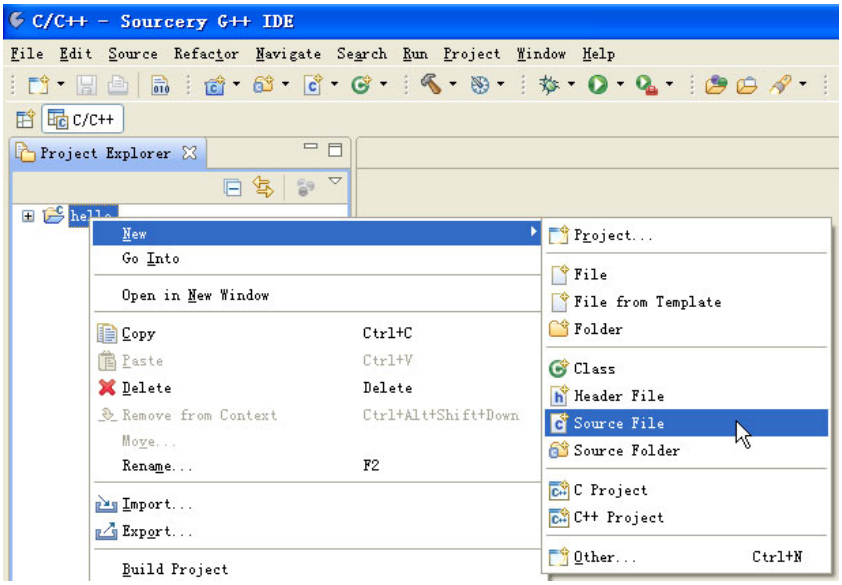
建立工程文件

2、在 **Project name:**中填入 **hello**，然后选择 **Finish** 完成工程建立，如下图所示。



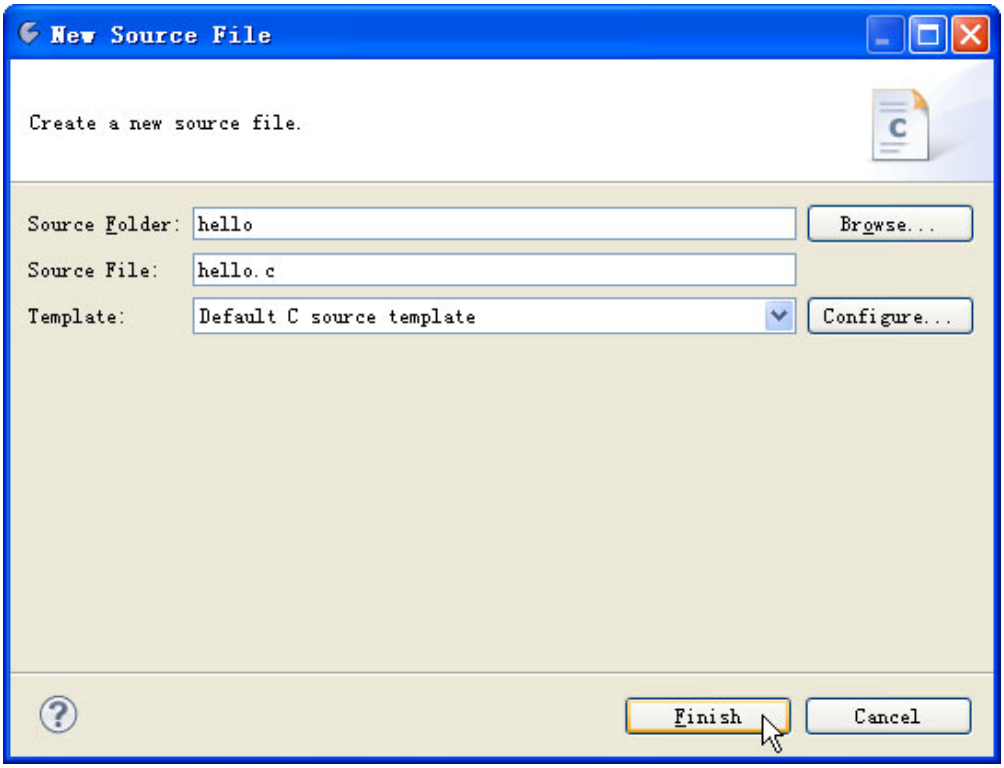
选择 Toolchains

3、在 Project Explorer 中建好的工程文件名 hello 上点右键,选择 **New -> Source File**，如下图所示。



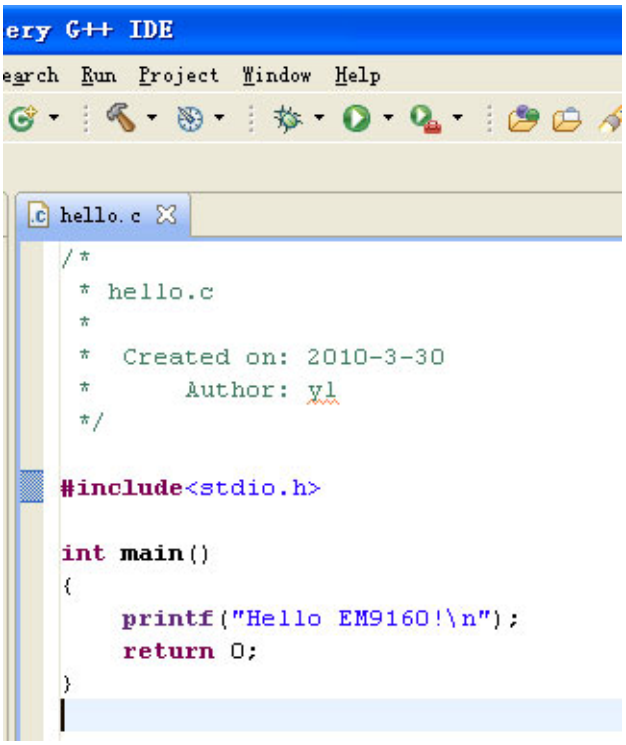
建立源程序文件

4、源程序文件命名为 **hello.c**，如下图所示。



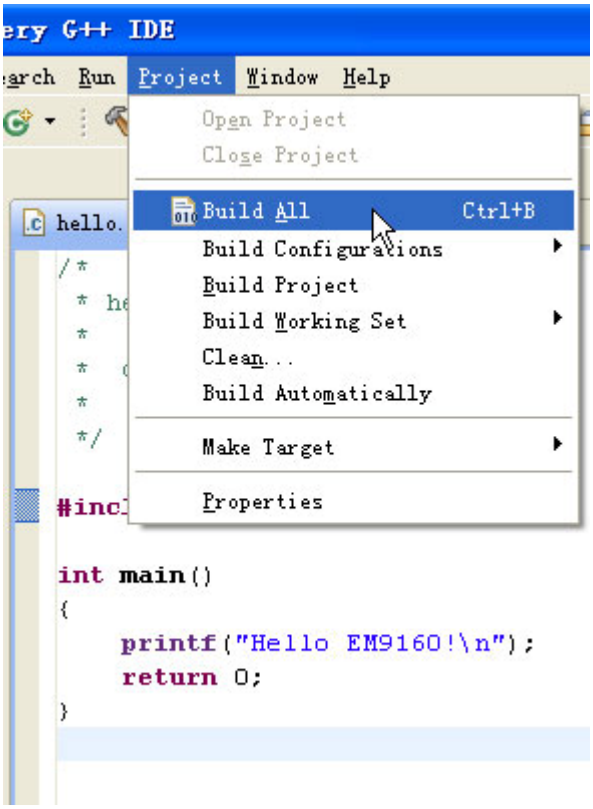
为源程序文件命名

5、系统自动建立源程序文件，输入程序源码并保存，如下图所示。



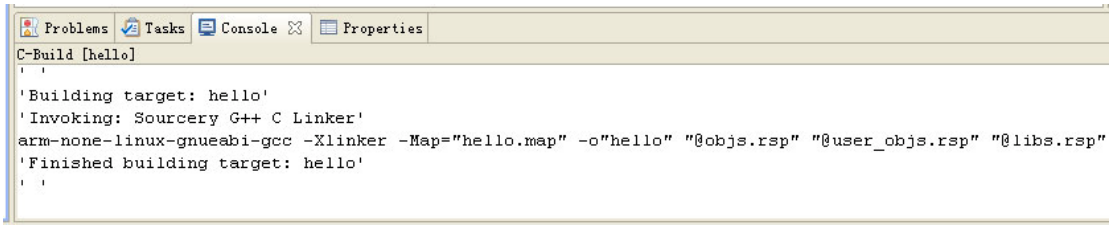
输入程序源码

6、选择 **Project -> Build All** 编译源码，如下图所示。



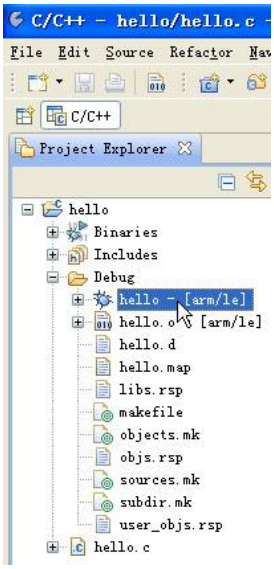
编译源码

7、编译完成，底部区域会显示出编译信息，如下图所示。如果程序有错误，也会在底部区域提示。



编译信息

此时，通过左侧 Project Explorer 可以看到已经生成应用程序 hello，如下图所示。

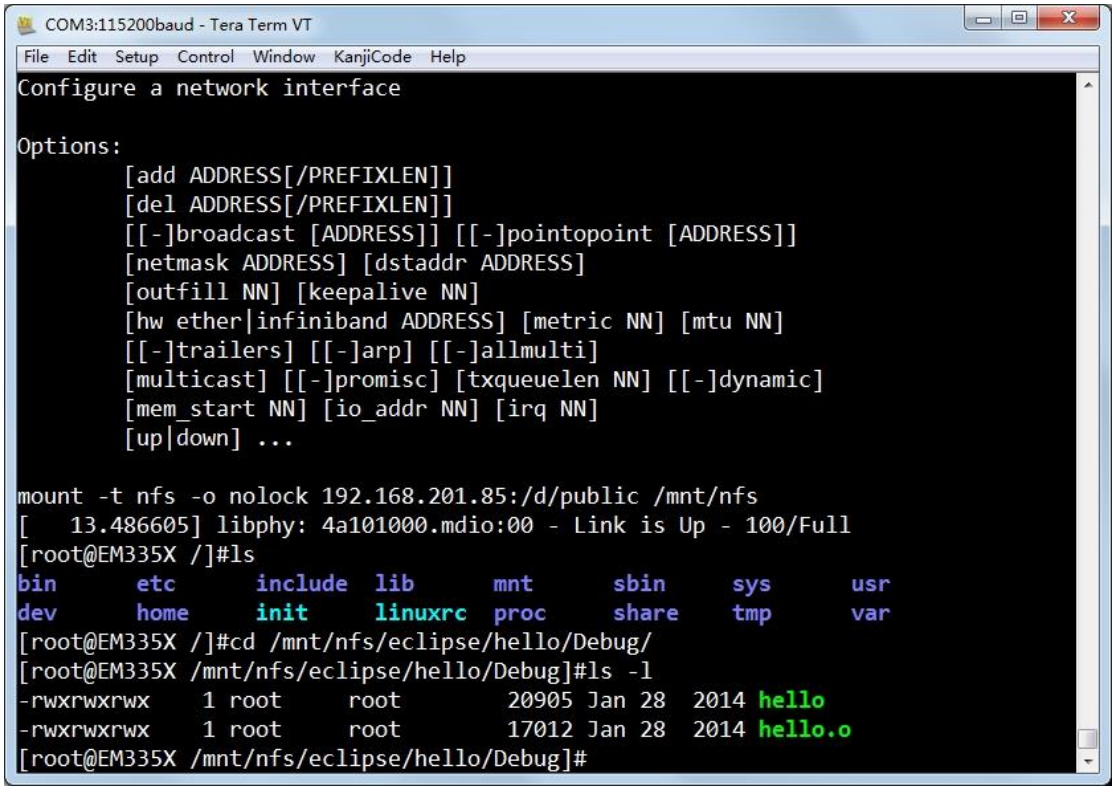


生成应用程序

8、工程文件默认保存地址是 **D:\public\eclipse**；而挂载到 ES6210 下的开发主机文件夹为 **D:\public**，在超级终端通过如下命令进入工程文件夹：

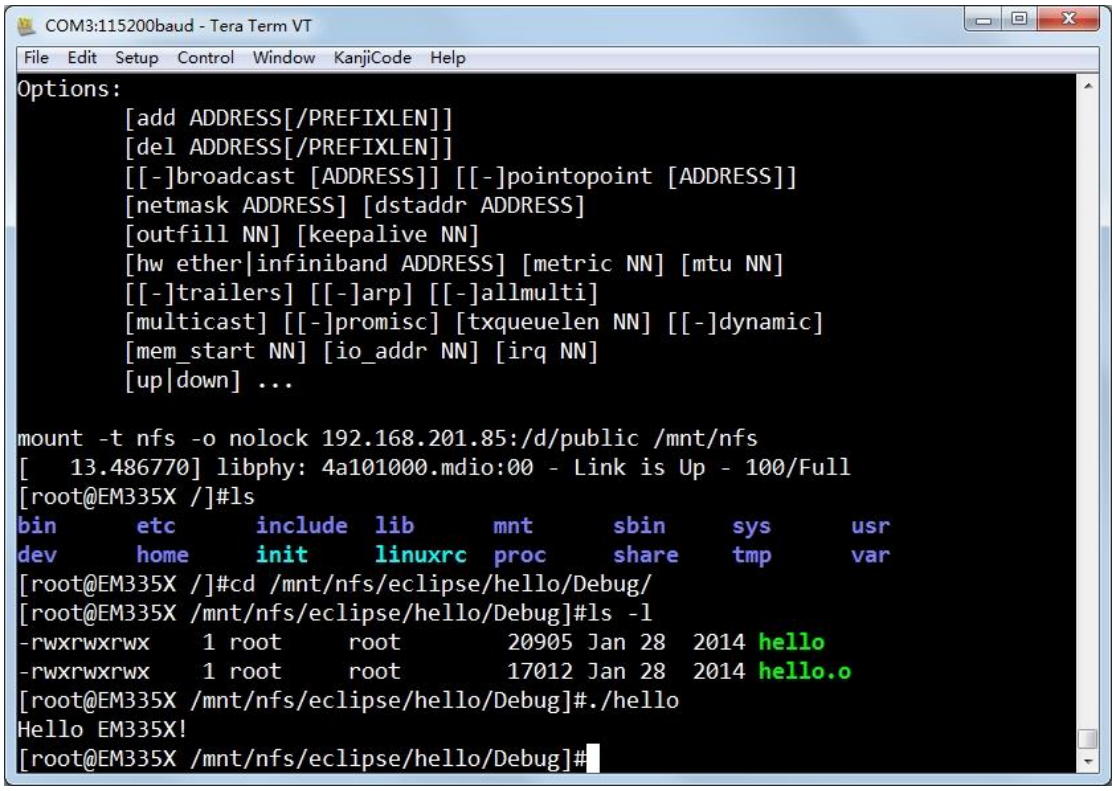
cd /mnt/nfs/eclipse/hello/Debug

9、通过超级终端使用命令 **ls -l** 查看挂载文件夹，可以看到 hello，如下图所示。



通过超级终端查看挂载文件夹

10、使用命令**./hello** 执行应用程序，运行成功，终端显示出打印结果，如下图所示。



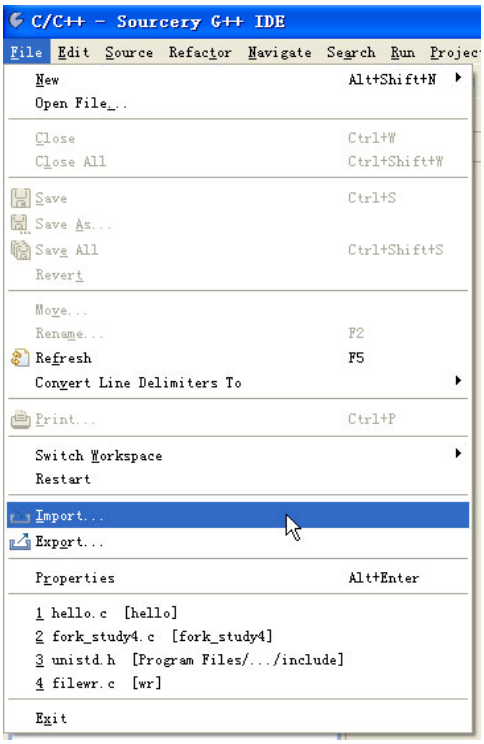
执行应用程序

这样，用户就完成了从工程文件建立到程序运行测试的一个完整过程。

4.2 打开已有的工程文件 wr

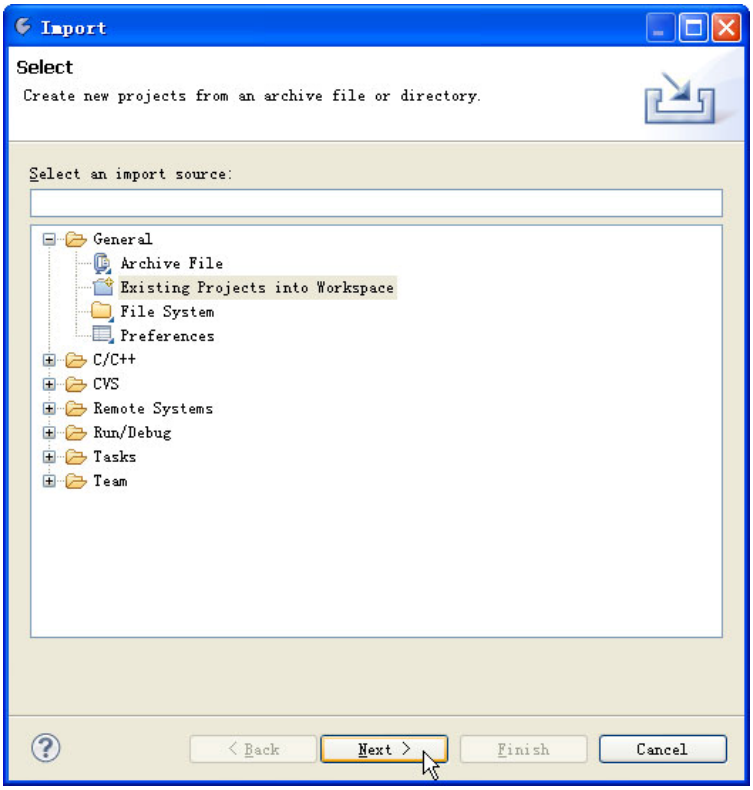
1、在英创公司的开发光盘中找到 wr 文件夹，复制到开发主机任意地址。

2、启动 eclipse，选择 **File -> Import…**，如下图所示。



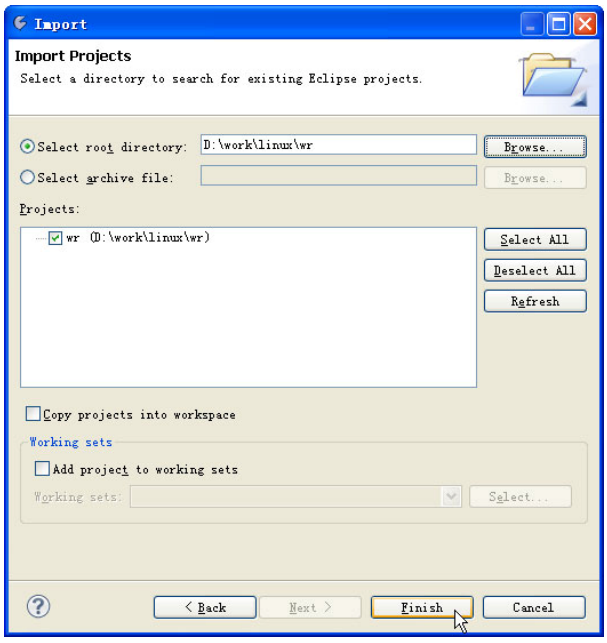
载入工程文件

3、选择 **General -> Existing Projects into Workspace**，如下图所示。



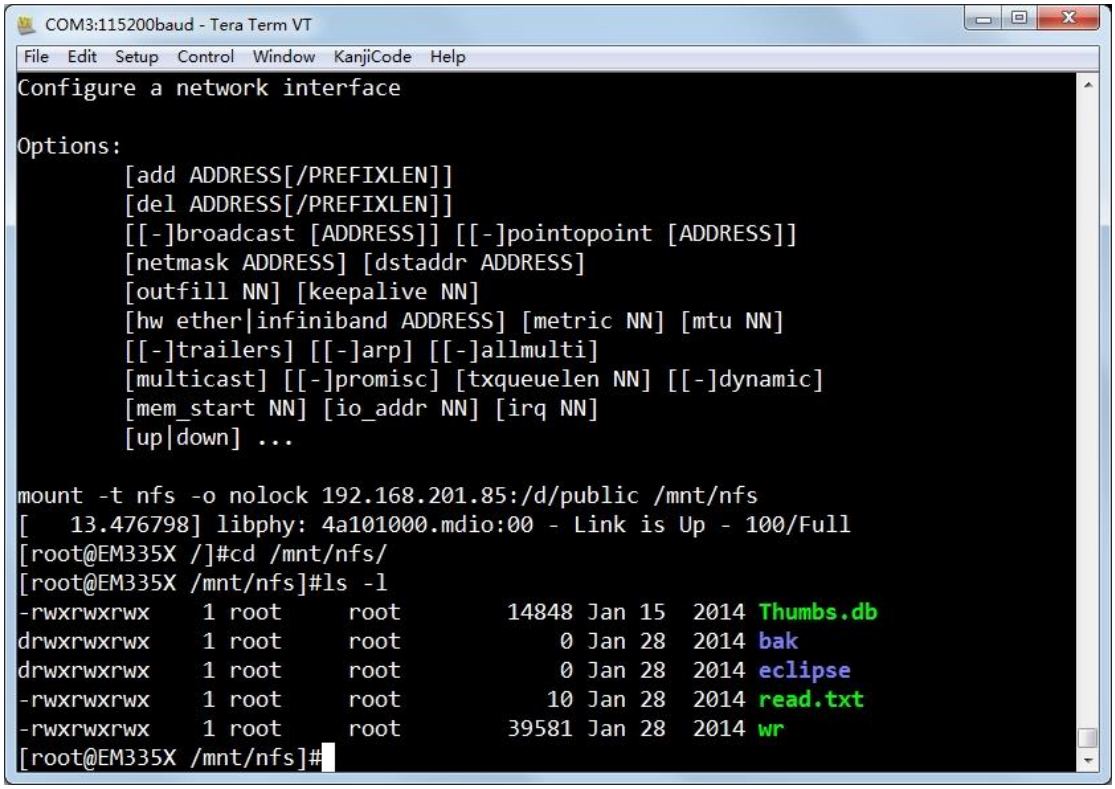
载入已存在的工程文件

4、通过**Select root directory:**右边的**Browse…**按钮找到之前复制到开发主机中的wr文件夹，其余选项保持默认设置，如下图所示。



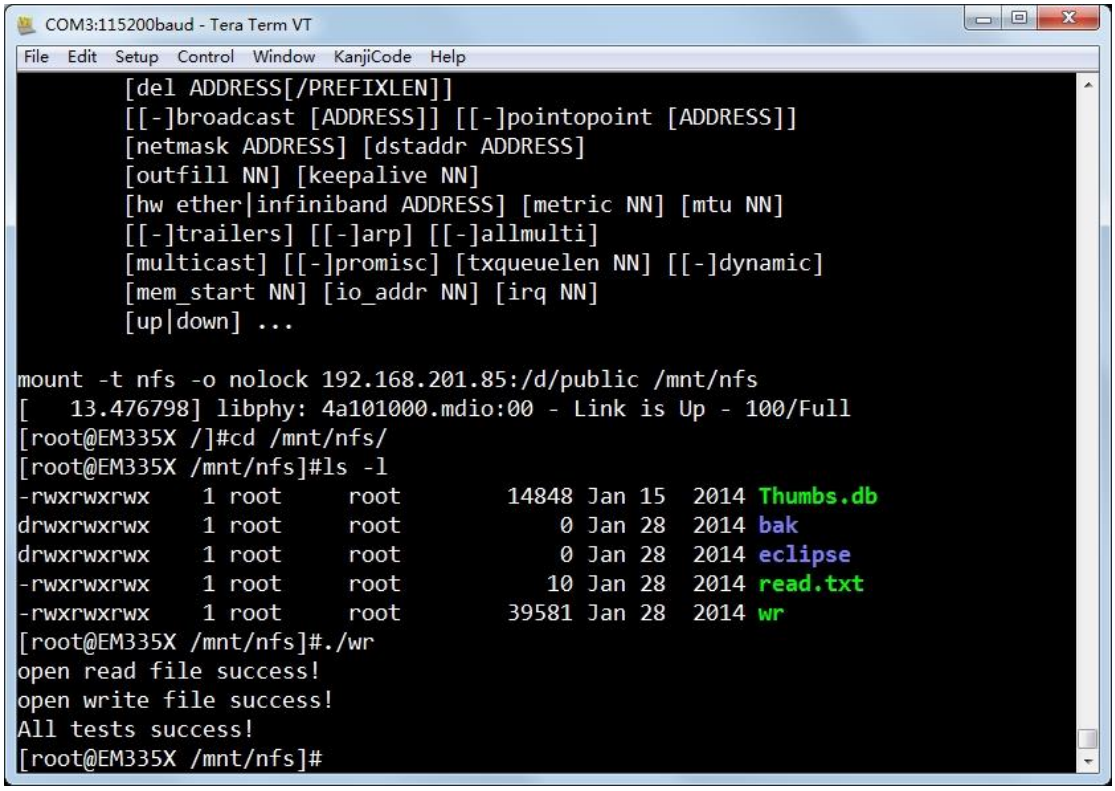
找到wr文件夹

5、打开工程文件以后，如3.1节的步骤，进行编译，然后通过超级终端进入工程文件夹。这个程序是要打开一个名为“read.txt”的文件，将其内容读出然后复制到另一个名为“write.txt”的文件中，因此需将**read.txt**（这个文件放在**wr\Debug**中）也复制到**D:\public**下。之后通过超级终端查看该文件夹，可以看到wr和read.txt，如下图所示。



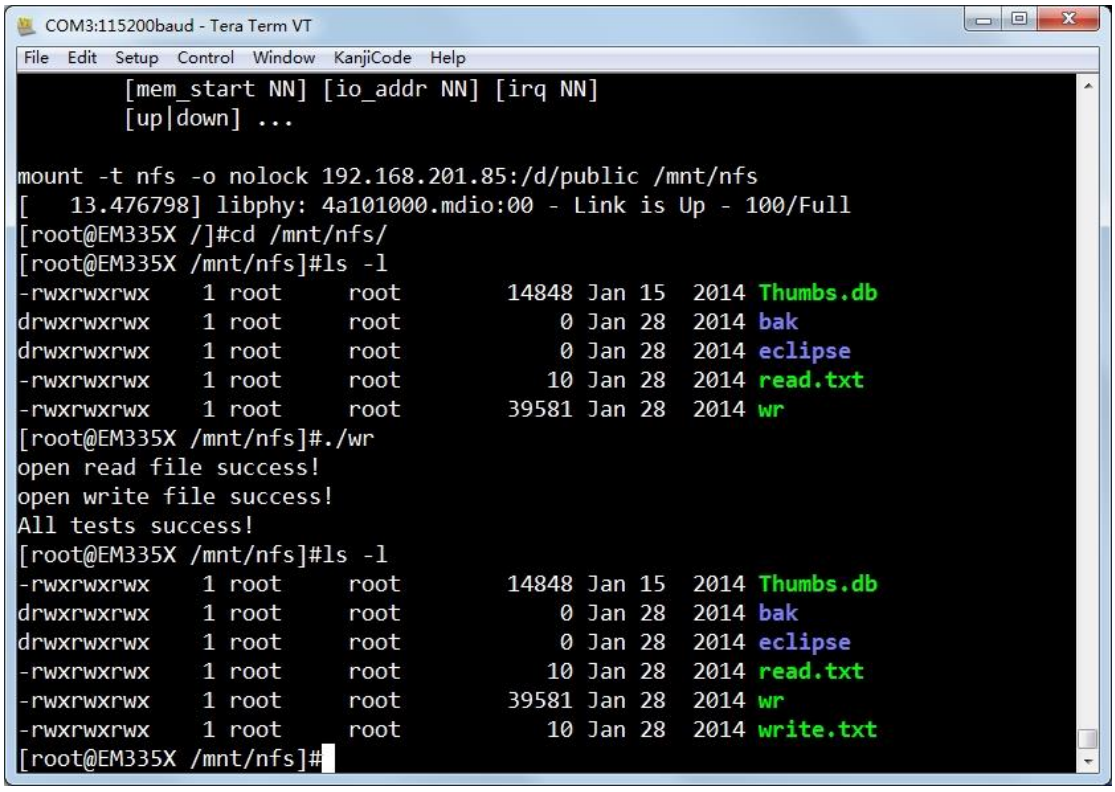
通过超级终端查看复制进来的两个文件

6、使用命令**./wr**执行应用程序，运行成功，终端显示出打印结果，如下图所示。



执行应用程序

7、再次使用**ls -l**命令查看，程序已经新建了write.txt文件，如下图所示。在开发主机中打开该文件和read.txt对比，确认内容一致，表明程序运行成功。



检查程序运行结果

这样，用户就可以在任何一台已经搭建好开发平台的 PC 上编辑自己的应用程序，并在 ES6210 上进行运行测试了。

5 应用程序编程范例之一：串口通讯

ES6210 提供了 2 个标准异步串口：ttyS1、ttyS2，其中 ttyS1、ttyS2 均与 GPIO 的管脚复用，每个串口都有独立的中断模式，使得多个串口能够同时实时进行数据收发。各个串口的驱动已经包含在 Linux 操作系统的内核中，ES6210 在 Linux 系统启动完成时，各个串口已作为字符设备完成了注册加载，用户的应用程序可以以操作文件的方式对串口进行读写，从而实现数据收发的功能。

5.1 串口编程接口函数

在 Linux 中，所有的设备文件都位于“/dev”目录下，ES6210 上六个串口所对应的设备名依次为：“/dev/ttyS1”、“/dev/ttyS2”。ttyS1、ttyS2 均为高速串口，其波特率可达 3Mbps，数据位为 8-bit，支持奇偶校验、MARK / SPACE 设置。

在 Linux 下操作设备的方式和操作文件的方式是一样的，调用 open( )打开设备文件，再调用 read( )、write( )对串口进行数据读写操作。这里需要注意的是打开串口除了设置普通的读写之外，还需要设置 O_NOCTTY 和 O_NDLEAY，以避免该串口成为一个控制终端，有可能会影响到用户的进程。如：

```
sprintf( portname, "/dev/ttyS%d", PortNo );              //PortNo为串口端口号，从1开始
m_fd = open( portname,O_RDWR | O_NOCTTY | O_NONBLOCK);
```

作为串口通讯还需要一些通讯参数的配置，包括波特率、数据位、停止位、校验位等参数。在实际的操作中，主要是通过设置 struct termios 结构体的各个成员值来实现，一般会用到的函数包括：

```
tcgetattr( ) ;
tcflush( );
cfsetispeed( );
cfsetospeed( );
tcsetattr( );
```

其中各个函数的具体使用方法这里就不一一介绍了，用户可以参考 Linux 应用程序开发的相关书籍，也可参看 Step2_SerialTest 中 Serial.cpp 模块中 set_port( )函数代码。

5.2 串口综合应用示例

Step2 _SerialTest 是一个支持异步串口数据通讯的示例，该例程采用了面向对象的 C++编程，把串口数据通讯作为一个对象进行封装，用户调用该对象提供的接口函数即可方便地完成串口通讯的操作。

CSerial 类介绍

利用上一小节中介绍的串口 API 函数，封装了一个支持异步读写的串口类 CSerial，CSerial 类中提供了 4 个公共函数、一个串口数据接收线程以及数据接收用到的数据 Buffer。

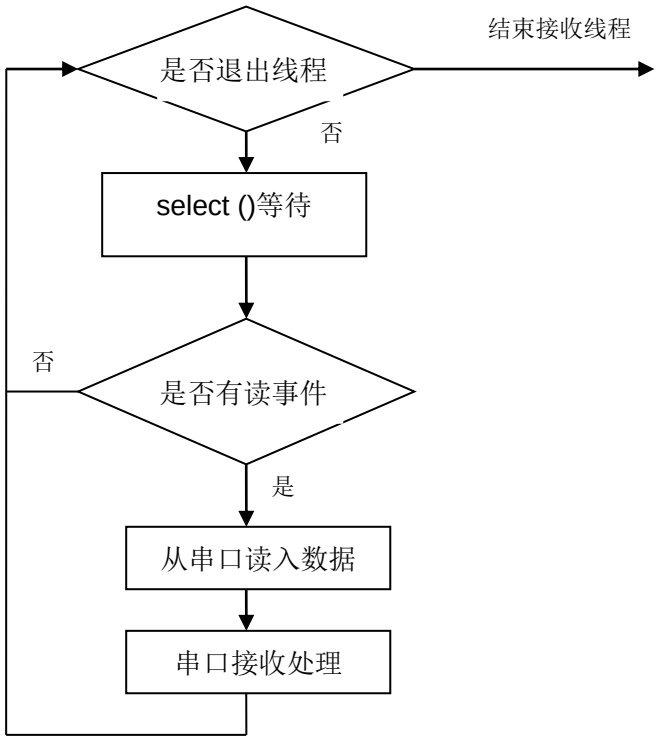
```
class CSerial
{
private:
    //通讯线程标识符ID
    pthread_t  m_thread;
    // 串口数据接收线程
    static int ReceiveThreadFunc( void* lparam );
public:
    CSerial();
    virtual ~CSerial();

    int          m_fd;                  // 已打开的串口文件描述符
    int     m_DatLen;
    char         DatBuf[1500];
    int     m_ExitThreadFlag;

    // 按照指定的串口参数打开串口，并创建串口接收线程
    int OpenPort( int PortNo, int baudrate, char databits, char stopbits, char parity );
    // 关闭串口并释放相关资源
    int ClosePort( );
    // 向串口写数据
    int WritePort( char* Buf, int len );
    // 接收串口数据处理函数
    virtual int PackagePro( char* Buf, int len );
};
```

OpenPort 函数用于根据输入串口参数打开串口，并创建串口数据接收线程。在 Linux 环境中是通过函数 pthread_create( )创建线程，通过函数 pthread_exit( )退出线程。Linux

线程属性存在有非分离（缺省）和分离两种，在非分离情况下，当一个线程结束时，它所占用的系统资源并没有被释放，也就是没有真正的终止；只有调用 pthread_join( )函数返回时，创建的线程才能释放自己占有的资源。在分离属性下，一个线程结束时立即释放所占用的系统资源。基于这个原因，在我们提供的例程中通过相关函数将数据接收线程的属性设置为分离属性。如：

```
// 设置线程绑定属性
res = pthread_attr_setscope( &attr, PTHREAD_SCOPE_SYSTEM );
// 设置线程分离属性
res += pthread_attr_setdetachstate( &attr, THREAD_CREATE_DETACHED );
```

ReceiveThreadFunc 函数是串口数据接收和处理的主要核心代码，在该函数中调用 select( )，阻塞等待串口数据的到来。对于接收到的数据处理也是在该函数中实现，在本例程中处理为简单的数据回发，用户可结合实际的应用修改此处代码，修改 PackagePro( )函数即可。流程如下：




```
int CSerial::ReceiveThreadFunc(void* lparam)
{
    CSerial *pSer = (CSerial*)lparam;

    //定义读事件集合
    fd_set fdRead;
```

```
int ret;
    struct timeval    aTime;

    while( 1 )
    {
        //收到退出事件，结束线程
        if( pSer->m_ExitThreadFlag )
        {
            break;
        }
        FD_ZERO(&fdRead);
        FD_SET(pSer->m_fd,&fdRead);
        aTime.tv_sec = 0;
        aTime.tv_usec = 300000;
        ret = select( pSer->m_fd+1,&fdRead,NULL,NULL,&aTime );
        if (ret < 0 )
        {
            //关闭串口
            pSer->ClosePort( );
            break;
        }
        if (ret > 0)
        {
            //判断是否读事件
            if (FD_ISSET(pSer->m_fd,&fdRead))
            {
                //data available, so get it!
                pSer->m_DatLen = read( pSer->m_fd, pSer->DatBuf, 1500 );
                // 对接收的数据进行处理，这里为简单的数据回发
                if( pSer->m_DatLen > 0 )
                {
                    pSer->PackagePro( pSer->DatBuf, pSer->m_DatLen);
                }
                // 处理完毕
            }
        }
    }
    printf( "ReceiveThreadFunc finished\n");
    pthread_exit( NULL );
    return 0;
}
```

需要注意的是，select( )函数中的时间参数在 Linux 下，每次都需要重新赋值，否则会自动归 0。

CSerial 类的实现代码请参见 Serial.CPP 文件。

CSerial 类的调用

CSerial 类的具体使用也比较简单，主要是对于类中定义的 4 个公共函数的调用，以下为 Step2_SerialTest.cpp 中相关代码。

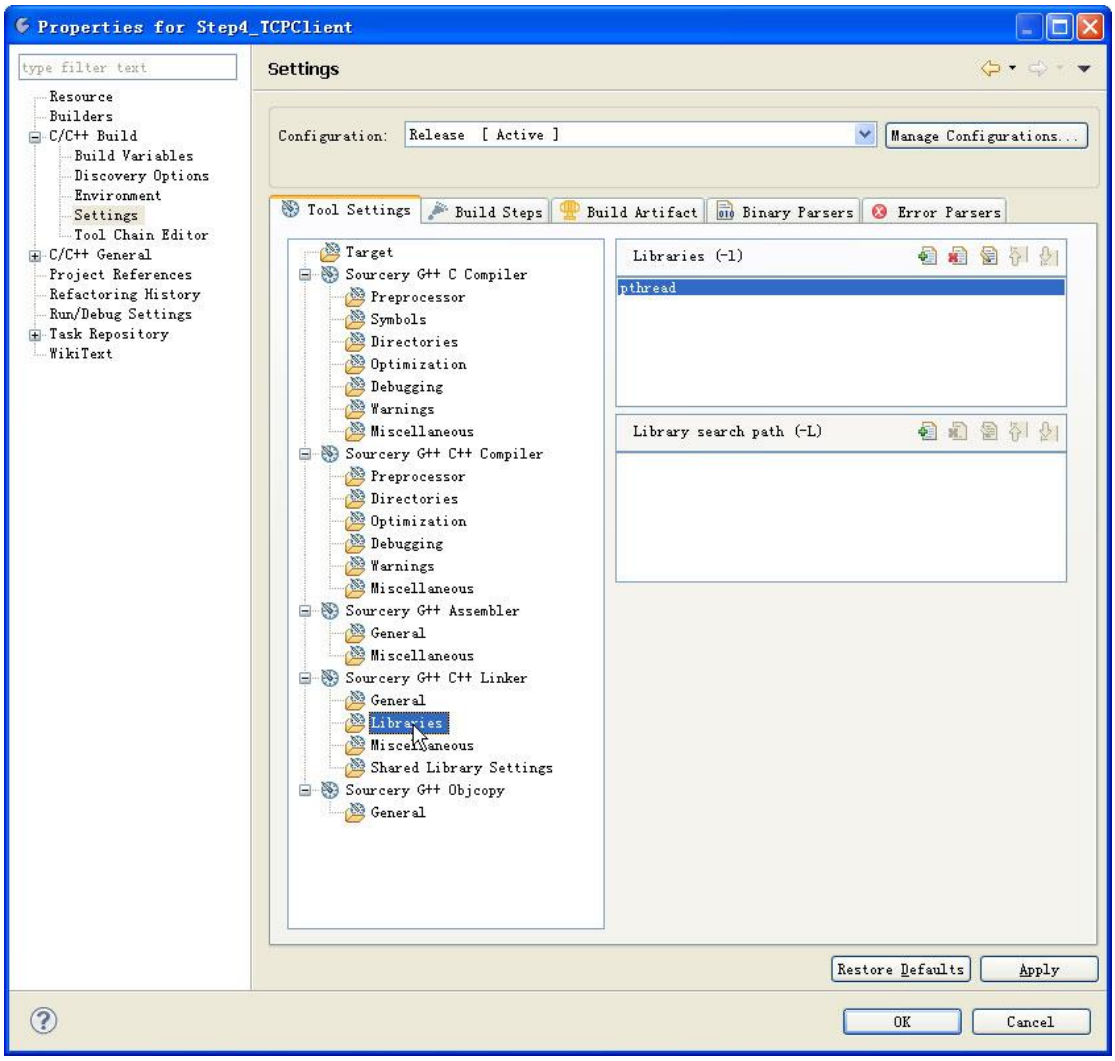
```
class CSerial    m_Serial;
int main( int argc,char* argv[] )
{
    int         i1;
    int          portno,    baudRate;
    char        cmdline[256];

   printf( "Step2_SerialTest V1.0\n" );
   // 解析命令行参数：串口号    波特率
   if( argc > 1 )          strcpy( cmdline, argv[1] );
   else                            portno = 1;
   if( argc > 2 )
   {
          strcat( cmdline, " " );
          strcat( cmdline, argv[2] );
          scanf( cmdline, "%d %d", &portno, &baudRate );
   }
   else
   {
          baudRate = 115200;
   }
   printf( "port:%d baudrate:%d\n", portno, baudRate);
   //打开串口相应地启动了串口数据接收线程
   i1 = m_Serial.OpenPort( portno, baudRate, '8', '1', 'N');
   if( i1<0 )
   {
          printf( "serial open fail\n");
          return -1;
   }
   //进入主循环，这里每隔1s输出一个提示信息
   for( i1=0; i1<10000;i1++)
   {
          sleep(1);
          printf( "%d \n", i1+1);
   }
```

```
    m_Serial.ClosePort( );
    return 0;
}
```

从上面的代码可以看出，程序的主循环只需要实现一些管理性的功能，在本例程中仅仅是每隔 1s 输出一个提示信息，在实际的应用中，可以把一些定时查询状态的操作、看门狗的喂狗等操作放在主循环中，这样充分利用了 Linux 多任务的编程优势，利用内核的任务调度机制，将各个应用功能模块化，以便于程序的设计和管理。这里顺便再提一下，在进行多个串口编程时，也可以利用本例程中的 CSerial 类为基类，根据应用需求派生多个 CSerial 派生类实例，每一个派生类只是重新实现虚函数 PackagePro(…)，这样每个串口都具有一个独立的串口数据处理线程，利用 Linux 内核的任务调度机制以实现多串口通讯功能。

Step2_SerialTest 的编译设置

在该例程中用到了线程操作函数，由于线程库不是缺省库，eclipse 编译可以通过，但是 link 会出错，需要配置 eclipse IDE 的编译参数，Linker 链接中增加线程库。在 Project Explorer 视窗下，选择 Step2_SerialTest 工程文件，然后点击鼠标右键，选择 **Properties** 项，在窗口中选择 **C/C++ Build -> Settings -> Tool Settings -> Sourcery G++ C++ Linker -> Libraries**，如下图所示。其中的一个窗口用于指定库文件的名称，一个用于指定库文件的路径，对于系统中已有的线程库 lpthread 文件，就不需要指定路径。



链接库文件

6 应用程序编程范例之二：TCP 服务器

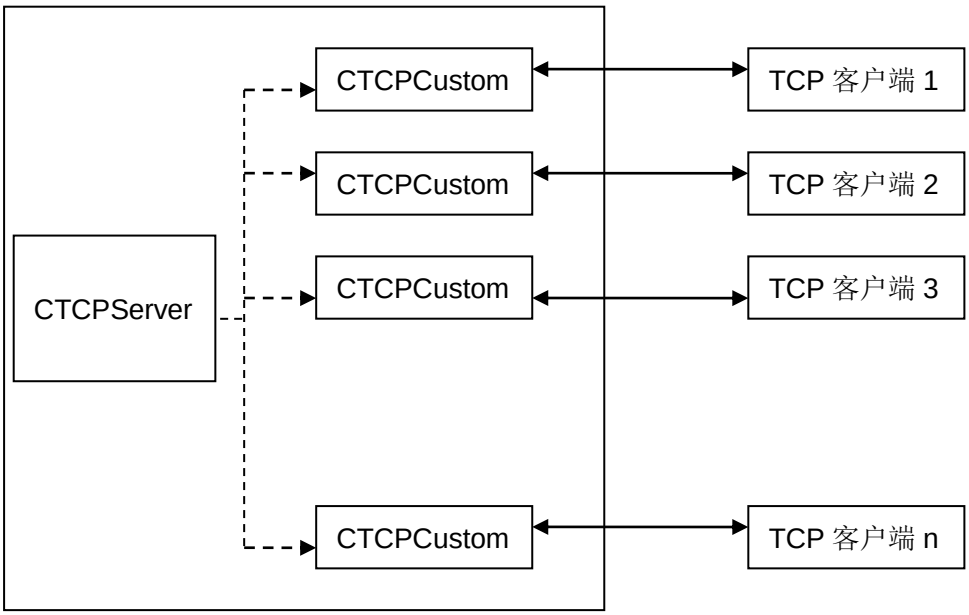
以太网在嵌入式领域的应用非常广泛，在工控领域中比较常见的就是利用 TCP/IP 协议进行数据通讯。ES6210 具有 10M/100M 自适应网络接口，非常适用于作为网络应用的开发平台。在网络应用中，编程显得尤为重要，在本章主要介绍 ES6210 作为 TCP 服务器方式的应用——支持多连接的 TCP 服务器示例程序：Step3_TCPServer。

6.1 TCP Socket 编程

在进行网络应用程序开发方面大多是采用套接字 Socket 技术，Linux 的系统平台上也是如此。Socket 编程的基本函数有 socket( )、bind( )、listen( )、accept( )、send( )、sendto( )、recv( )、recvfrom( )、connect( )等，各个函数的具体使用方法这里就不一一介绍了，用户可以参考 Linux 应用程序开发的相关书籍。

6.2 支持多连接的 TCP 服务器应用示例

Step3_TCPServe 是一个支持多个客户端的连接 TCPServer 示例，该例程采用了面向对象的 C++编程，创建了 CTCPServer 和 CTCPCustom 两个类，其中 CTCPServer 类负责侦听客户端的连接，一旦有客户端请求连接，它就负责接受此连接，并创建一个新的 CTCPCustom 类对象与客户端进行通讯，然后 CTCPServer 类接着监听客户端的连接请求，其流程如下：



CTCPServer 类

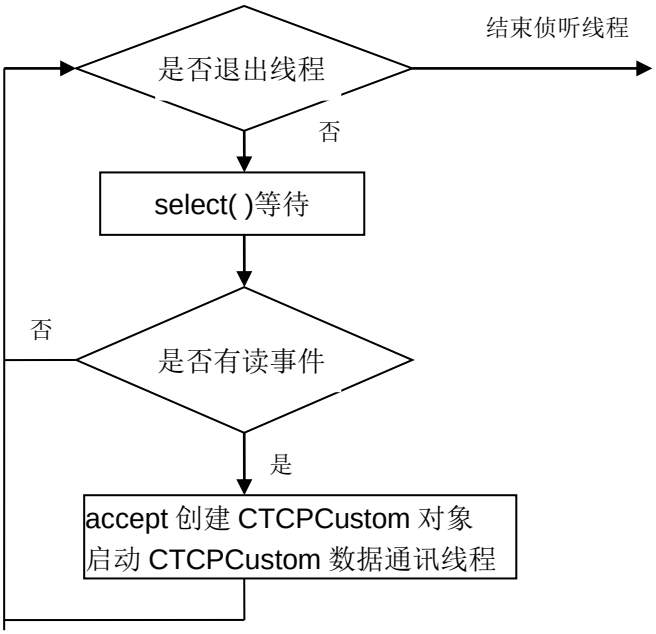
CTCPServer 类定义在 TCPServer.h 文件下，该类提供了 3 个公共函数，以及一个 Socket 侦听线程，公共的函数中 Open( )、Close( )用于启动或是关闭 TCP 服务。

```
class CTCPServer
{
private:
    pthread_t   m_thread;                          // 通讯线程标识符ID
    // Socket侦听线程
    static int SocketListenThread( void*lparam );
public:
    int             m_sockfd;                      // TCP服务监听socket
    int             m_ExitThreadFlag;
    int             m_LocalPort;                   // 设置服务端口号
    CTCPServer();
    virtual ~CTCPServer();
    int Open();                                    // 打开TCP服务
    int Close();                                   // 关闭TCP服务
    // 删除一个客户端对象连接 释放资源
    int RemoveClientSocketObject( void* lparam );
};
```

在Open( )函数中实现了打开套接字，将套接字设置为侦听套接字，并创建侦听客户端连接线程。在Linux应用程序中创建线程的方法在“5.2 串口综合应用示例”中有相关的说明，

在该例程中也是采取的同样方式。

SocketListenThread函数中调用select( )侦听客户端的TCP连接，流程如下：



同样的需要注意的是，select( )函数中的时间参数在 Linux 下每次都需要重新赋值，否则会自动归 0。CTCPServer 类的实现代码请参见 TCPServer.CPP 文件。

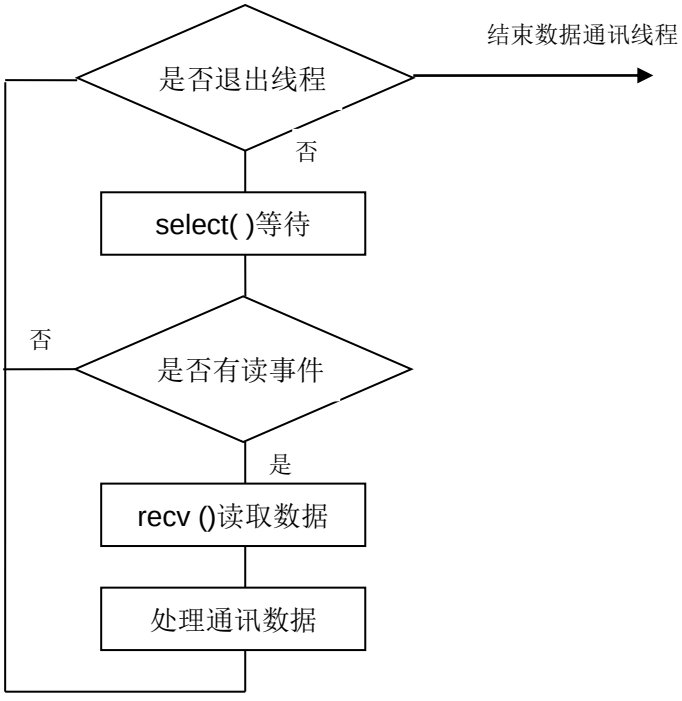
CTCPCustom类

CTCPCustom的定义在TCPCustom.h文件下。

```
class CTCPCustom
{
public:
	CTCPCustom();
	virtual ~CTCPCustom();
public:
	char	m_RemoteHost[100];			// 远程主机IP地址
	int	m_RemotePort;				// 远程主机端口号
	int	m_socketfd;				// 通讯socket
	int	m_SocketEnable;
	int	m_ExitThreadFlag;
	CTCPServer*	m_pTCPServer;
private:
```

```
        // 通讯线程函数
        pthread_t   m_thread;                                   // 通讯线程标识符ID
        static void* SocketDataThread(void* lparam);        // TCP连接数据通讯线程
    public:
        int RecvLen;
        char RecvBuf[1500];
        // 打开socket，创建通讯线程
        int Open(void* lparam);
        // 关闭socket，关闭线程，释放Socket资源
        int Close();
       // 向客户端发送数据
        int SendData(const char * buf , int len );
    };
```

其中的 SocketDataThread 函数是实现 TCP 连接数据通讯的核心代码，在该函数中调用 select( )等待 TCP 连接的通讯数据，对于接收的 TCP 连接数据的处理也是在该函数中实现，在本例程中处理为简单的数据回发，用户可结合实际的应用修改此处代码，流程如下：



CTCPServer 类的调用

CTCPSerer 类的具体使用也比较简单，主要是调用对于类中定义 Open 函数来启动各个 TCP 通讯线程，反而在主循环中需要实现的功能代码不多了，在本例程中仅仅为每隔 1s 输出提示信息。以下为 Step3_TCPServer.cpp 中的相关代码。

```
class CTCPServer    m_TCPServer;
int main()
{
        int i1;
        printf( "Step3_TCPTest V1.0\n" );
        // 给TCP服务器端口赋值
        m_TCPServer.m_LocalPort = 1001;
        // 创建Socket，启动TCP服务器侦听线程
        i1 = m_TCPServer.Open( );
        if( i1<0 )
        {
                printf( "TCP Server start fail\n");
                return -1;
        }
        // 进入主循环，主要是负责管理工作
        for( i1=0; i1<10000;i1++)                    // 实际应用时，可设置为无限循环
        {
                sleep(1);
                printf( "%d \n", i1+1);
        }
        m_TCPServer.Close( );
        return 0;
}
```

Step3_TCPServer 的编译设置

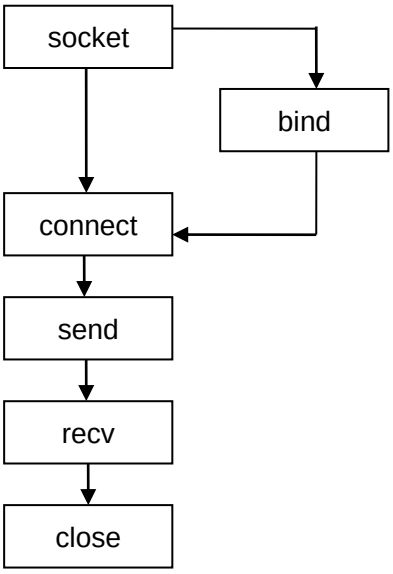
由于使用到 Linux 的线程功能，因此本实例与 Step2_SerialTest 实例程序一样，需要对缺省的编译设置进行修改，具体方法请参见“5.2　串口综合应用示例”相关内容。

7 应用程序编程范例之三：TCP 客户端

本章主要介绍 ES6210 作为 TCP 客户端方式的应用示例 Step4_TCPClient。

7.1 TCP 客户端 Socket 编程流程

在利用 Socket 进行 TCP 客户端编程时，建立 TCP 连接的过程一般比较简单，首先客户端调用 socket()函数建立流式套接字，然后调用 connect()函数请求服务器端建立 TCP 连接，成功建立连接后即可与服务器端进行 TCP/IP 数据通讯，流程如下：



7.2 TCPClient 应用示例

Step4_TCPClient 是一个具有自动管理功能的 TCP 客户端应用示例。作为 TCP 客户端主动和服务器端建立 TCP 连接的过程编程相对简单，直接调用相关的 Socket 函数即可，建立 TCP 连接的功能封装在 CTCPClient 类中。嵌入式的应用场合大多是处于长期运行无人值守的状态，可能会遇到需要一直保持 TCP 客户端连接的情况，Step4_TCPClient 例程基于这种需求，专门封装了一个 CTCPClientManager 管理类对 TCPClient 的连接进行自动管理，包括启动建立 TCP 的客户端连接、查询 TCP 连接的状态、添加多个 TCP 客户端连接等功能。

CTCPClient 类

CTCPClient 类定义在 TCPClient.H 文件下，该类提供了 4 个公共函数，以及一个数据通讯线程，调用该类中的相关函数与 TCP 服务器端建立连接。

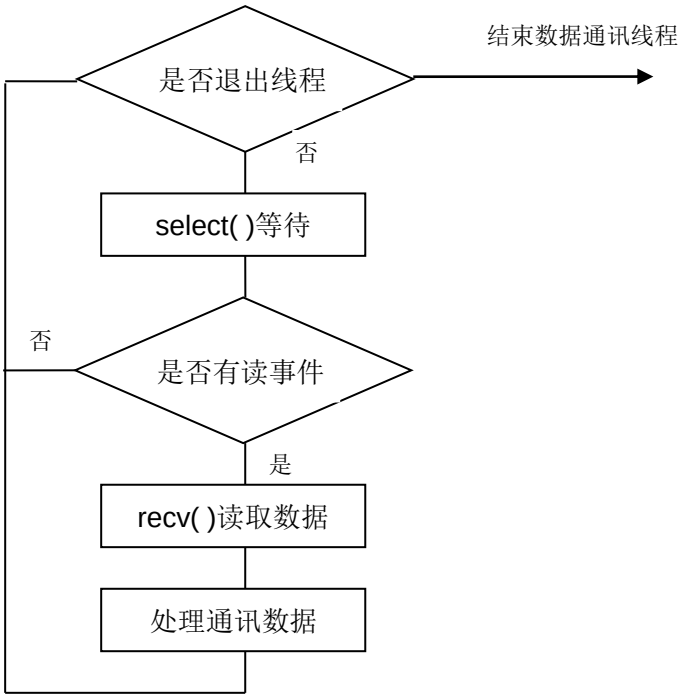
```
class CTCPClient
{
private:
        pthread_t   m_thread;                                   // 通讯线程标识符ID
        // 数据通讯处理线程函数
        static int SocketThreadFunc( void*lparam );
public:
        // TCP通讯Socket
        int                 m_sockfd;
        int                 m_sockclose;
        int                 m_ExitThreadFlag;
        // 远程主机IP地址
        char        m_remoteHost[255];
        // 远程主机端口
       int          m_port;
       char RecvBuf[1500];
        int    m_nRecvLen;

public:
        CTCPClient();
        virtual ~CTCPClient( );
        // 打开创建客户端socket
        int Open( char* ServerIP, int ServerPort );
        // 关闭客户端socket
        int Close( );
        // 与服务器端建立连接  并创建数据通讯处理线程
        int Connect();
        // 向服务器端发送数据
        int SendData( char * buf , int len);
};
```

Open 函数执行创建打开 socket 操作，并设置远端 TCP 服务器的 IP 和端口。

Connect 函数调用 connect( )与远端 TCP 服务器建立连接，调用 select( )等待 TCP 连接的建立，TCP 连接建立成功，则创建 TCP 数据通讯处理线程。

SocketThreadFunc 函数是实现 TCP 连接数据通讯的核心代码，在该函数中调用

select( )，等待 TCP 连接的通讯数据，对于接收的 TCP 连接数据的处理也是在该函数中实现，在本例程中处理为简单的数据回发，用户可结合实际的应用修改此处代码。流程如下：



CTCPClientManager 类

TCP 客户端连接定义为四个状态：

```
enum   CONNSTATE{ csWAIT, csINIT, csCLOSED, csOPEN };
```

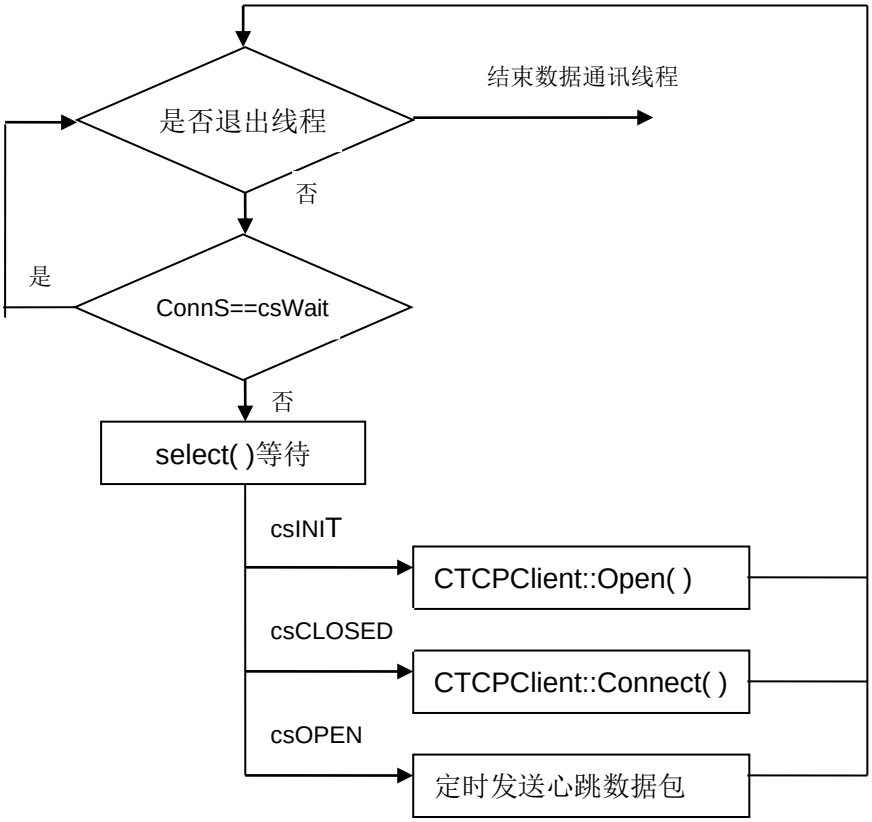
其中 csOPEN 表明 TCP 客户端连接建立。

CTCPClientManager 所封装的功能函数就是根据连接的各个状态对 TCP 客户端连接进行管理，CTCPClientManager 类定义在 TCPClientManager.H 文件下。

```
class CTCPClientManager
{
private:
    // TCPClient连接管理线程
    static int TCPClientThreadFunc( void* lparam );
public:
    TCPCLIENT_INFO          m_TCPClientInfo[TCPCLIENT_MAX_NUM];
    pthread_t               m_thread[TCPCLIENT_MAX_NUM];
```

```
        int                                 m_nTCPClientNum;
    public:
        CTCPClientManager( );
        ~CTCPClientManager();
        // 添加TCP客户端连接对象，输入参数为TCP服务器的IP和端口
        int AddTCPClientObject( char* pHostIP, int nHostPort );
        // 删除所有TCP客户端连接对象
        int DeleteAllTCPClient( );
        // 设置TCP客户端连接对象为csINIT状态
        int Open( int Idx );
        // 获取TCP客户端连接状态
        int     GetTCPClientState( int Idx );
        // 启动TCPClient连接管理操作，并创建TCPClient连接管理线程
        int Start( );
        // 关闭TCPClient连接管理操作
        int Stop( );
    };
```

TCPClientThreadFunc 函数是实现对 TCP 连接状态管理操作的核心代码，由于 Linux 下 sleep 的最小单位为秒，对于毫秒级的延时等待，在该函数中利用调用 select( )设置相关的时间参数来实现。流程如下：



CTCPClientManager 类的调用

CTCPClientManager 类的具体使用过程：首先调用类中定义 AddTCPClientObject 加载 TCP 连接对象，然后调用类中定义 Start 函数来启动 TCP 连接自动管理线程，Open 函数将 TCP 连接状态设置为 csINIT 状态。本例程中主循环的操作为每隔 1s 查询 TCPClient 连接的状态，如果状态为 csWait，程序调用 Open 函数将其设置为 csINIT 状态，则 TCPC 连接管理线程将自动进行与 TCP 服务器端建立连接的操作。

以下为 Step4_TCPClient.cpp 中的相关代码。

```
class CTCPClientManager        TCPClntManager;
int main()
{
      int i1, i2, i3;
      // 添加一个TCP客户端连接对象
```

```
        TCPClntManager.AddTCPClientObject( "192.168.201.121", 1001 );
        // 启动TCPClient连接管理操作，并创建TCPClient连接管理线程
        TCPClntManager.Start();
        for( i1=0; i1<TCPClntManager.m_nTCPClientNum; i1++ )
        {
            // 设置TCP客户端连接初始状态，连接管理线程将自动进行TCP的连接操作
            TCPClntManager.Open(i1);
        }
        for(i1=0; i1<10000; i1++)
        {
            sleep(1);
            for( i2=0; i2<TCPClntManager.m_nTCPClientNum; i2++ )
            {
                // 查询TCP客户端连接状态
                i3 = TCPClntManager.GetTCPClientState(i2);
                printf( "TCP Connect%d State: %d \n", i2+1, i3 );
                if( i3==0 )
                {
                // 设置TCP客户端连接初始状态，连接管理线程将自动进行TCP连接操作
                    TCPClntManager.Open( i2 );
                }
            }
        }
        return 0;
}
```

8 应用程序编程范例之四：WiFi/蓝牙的应用

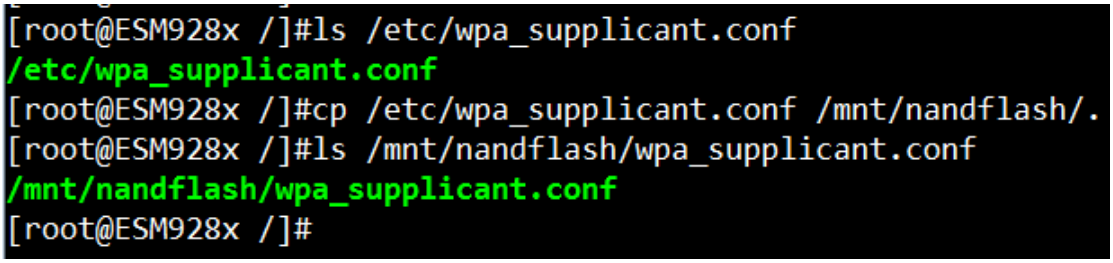
ES6210 上集成了模块 AP6210，将 WiFi 和蓝牙功能一同集成在核心板上，客户无需外扩模块和担心接口的稳定性等问题，底板也不需要额外的电路就能够使用 WiFi 和蓝牙的功能，十分适合工业现场应用，同时避免了硬件的设计，能够节省不少开发的周期。

8.1 Wi-Fi 的使用流程

1、首先需要设置 wpa_supplicant 的配置文件 wpa_supplicant.conf。该示例配置文件在目录/etc 下。

建议客户先把示例文件复制到/mnt/nandflash 中，再进行设置。如果出错还能在/etc 中找到示例配置文件:

[root@ ES6210/]#cp /etc/wpa_supplicant.conf /mnt/nandflash/.



Wi-Fi 配置文件

复制成功以后，进入 VI 模式编辑 wpa_supplicant.conf：

[root@ ES6210/]#vi /mnt/nandflash/wpa_supplicant.conf

进入 vi 模式可以看见 wpa_supplicant.conf 的配置，按“i”切换到插入模式进行编辑，客户只需修改其中的两项：

ssid=”Emtronix.20” //填入需要连接的 Wi-Fi 名称，本例为 Emtronix.20

psk=”0987654321” //填入连接 Wi-Fi 的密码，本例为 0987654321

修改完成以后，按“ESC”进入命令行模式，并在底行输入“:wq”（存盘退出），这样就设置完成。

```
[root@ESM928x /]#vi /mnt/nandflash/wpa_supplicant.conf

# WPA-PSK/TKIP

ctrl_interface=/var/run/wpa_supplicant

network={
        ssid="Emtronix.20"
        scan_ssid=1
        key_mgmt=WPA-EAP WPA-PSK IEEE8021X NONE
        pairwise=TKIP CCMP
        group=CCMP TKIP WEP104 WEP40
        psk="0987654321"
}
~
~
~
~
~
~
~
~
~
~
~
~
- /mnt/nandflash/wpa_supplicant.conf 1/12 8%
```

配置 ssid 和密码

2、加载无线模块的驱动：

[root@ ES6210/]#insmod /lib/modules/4.1.15/bcmdhd.ko \

firmware_path=/etc/firmware/ap6210/fw_bcmdhd.bin \

nvram_path=/etc/firmware/ap6210/bcmdhd.cal

加载成功之后，系统能检测到板上的无线模块。

```
[root@ESM928x /]#insmod /lib/modules/4.1.14/bcmdhd.ko firmware_path=/etc/fw_bcmd
hd.bin nvram_path=/etc/bcmdhd.cal
[  358.156061] dhd_module_init in
[  358.165403] Power-up adapter 'DHD generic adapter'
[  358.174516] wifi_platform_bus_enumerate device present 1
[  358.900487] mmc1: queuing unknown CIS tuple 0x80 (2 bytes)
[  358.907679] mmc1: queuing unknown CIS tuple 0x80 (3 bytes)
[  358.914831] mmc1: queuing unknown CIS tuple 0x80 (3 bytes)
[  358.923362] mmc1: queuing unknown CIS tuple 0x80 (7 bytes)
[  358.947796] mmc1: new high speed SDIO card at address 0001
[  359.009309] F1 signature OK, socitype:0x1 chip:0xa962 rev:0x1 pkg:0x9
[  359.018579] DHD: dongle ram size is set to 245760(orig 245760) at 0x0
[  359.027400] wifi_platform_get_mac_addr
[  359.031869] CFG80211-ERROR) wl_setup_wiphy : Registering Vendor80211)
[  359.044805] wl_create_event_handler(): thread:wl_event_handler:80 started
[  359.055616] CFG80211-ERROR) wl_event_handler : tsk Enter, tsk = 0xc67e13fc
[  359.063055] dhd_attach(): thread:dhd_watchdog_thread:81 started
[  359.069395] dhd_attach(): thread:dhd_dpc:82 started
[  359.074330] dhd_deferred_work_init: work queue initialized
[  359.273894] dhdsdio_write_vars: Download, Upload and compare of NVRAM succeeded.
[  359.536576] dhd_bus_init: enable 0x06, ready 0x06 (waited 0us)
[  359.546423] wifi_platform_get_mac_addr
[  359.554662] Firmware up: op_mode=0x0005, MAC=00:22:f4:a9:89:62
[  359.571444] dhd_preinit_ioctls buf_key_b4_m4 set failed -23
[  359.587801] Firmware version = wl0: Apr 30 2015 11:15:14 version 5.90.231 FWID 01-0
[  359.596667] dhd_preinit_ioctls wl ampdu_hostreorder failed -23
[  359.603410] dhd_wlfc_init(): successfully enabled bdcv2 tlv signaling, 79
[  359.611306] dhd_wlfc_init(): wlfc_mode=0x0, ret=-23
[  359.617351]
[  359.617351] Dongle Host Driver, version 1.141.88 (r)
[  359.617351] Compiled from
[  359.635060] Register interface [wlan0]  MAC: 00:22:f4:a9:89:62
[  359.635060]
[root@ESM928x /]#
```

加载 Wi-Fi 模块驱动

3、调用 wpa_supplicant 连接无线网：

[root@ES6210/]#wpa_supplicant -B -Dwext -iwlan0 -c /mnt/nandflash/wpa_supplicant.conf -d

参数说明：

-B 指定以守护进程模式运行，即程序将以后台模式运行。连接 Wi-Fi 需要 supplicant 一直运行，所以采用后台模式，不会影响客户其他程序的运行。

-D 指定使用的驱动，这里是无线网，所以用 wext。

-i 指定网卡。

-c 指定使用的配置文件，这里是我们之前设置好放在/mnt/nandflash 中的配置文件。

-d 添加调试信息。

这条指令调用成功之后，工控主板将成功连接上在 supplicant.conf 中设置的 Wi-Fi。

```
[root@ESM928x /]#insmod /lib/modules/4.1.14/bcmdhd.ko firmware_path=/etc/fw_bcmd
hd.bin nvram_path=/etc/bcmdhd.calbcmdhd.cal
[root@ESM928x /]#
[root@ESM928x /]#
[root@ESM928x /]#
[root@ESM928x /]#wpa_supplicant -B -Dwext -iwlan0 -c /mnt/nandflash/wpa_supplica
nt.conf -d
Initializing interface 'wlan0' conf '/mnt/nandflash/wpa_supplicant.conf' driver 'wext' ctrl_interface 'N/A' bridge 'N/A'
Configuration file '/mnt/nandflash/wpa_supplicant.conf' -> '/mnt/nandflash/wpa_supplicant.conf'
Reading configuration file '/mnt/nandflash/wpa_supplicant.conf'
ctrl_interface='/var/run/wpa_supplicant'
Priority group 0
   id=0 ssid='Emtronix.20'
WEXT: cfg80211-based driver detected
[  952.626036] CFG80211-ERROR) wl_update_wiphybands : error reading vhtmode (-23)
SIOCGIWRANGE: WE(compiled)=22 WE(source)=21 enc_capa=0xf
  capabilities: key_mgmt 0xf enc 0xf flags 0x0
netlink: Operstate: linkmode=1, operstate=5
Own MAC address: 00:22:f4:a9:89:62
wpa_driver_wext_set_key: alg=0 key_idx=0 set_tx=0 seq_len=0 key_len=0
wpa_driver_wext_set_key: alg=0 key_idx=1 set_tx=0 seq_len=0 key_len=0
wpa_driver_wext_set_key: alg=0 key_idx=2 set_tx=0 seq_len=0 key_len=0
wpa_driver_wext_set_key: alg=0 key_idx=3 set_tx=0 seq_len=0 key_len=0
wpa_driver_wext_set_countermeasures
RSN: flushing PMKID list in the driver
Setting scan request: 0 sec 100000 usec
EAPOL: SUPP_PAE entering state DISCONNECTED
EAPOL: Supplicant port status: Unauthorized
EAPOL: KEY_RX entering state NO_KEY_RECEIVE
EAPOL: SUPP_BE entering state INITIALIZE
EAP: EAP entering state DISABLED
EAPOL: Supplicant port status: Unauthorized
EAPOL: Supplicant port status: Unauthorized
Using existing control interface directory.
Added interface wlan0
Daemonize..
[root@ESM928x /]#[  973.159448] CFG80211-ERROR) wl_cfg80211_connect : Connecting withe0:91:f5:86:fa:1a channel (11) ssid "Emtronix.20", len (11)

[  973.385146] wl_bss_connect_done succeeded with e0:91:f5:86:fa:1a
[  973.456563] wl_bss_connect_done succeeded with e0:91:f5:86:fa:1a
```

连接 Wi-Fi

5、成功连接上 WiFi 之后，可以输入指令自动获取动态 IP：

[root@ ES6210/]#udhcpc –i wlan0

```
[root@ESM928x /]#udhcpc -iwlan0
udhcpc (v1.15.3) started
Setting IP address 0.0.0.0 on wlan0
Sending discover...
Sending select for 192.168.201.107...
Lease of 192.168.201.107 obtained, lease time 86400
Setting IP address 192.168.201.107 on wlan0
Deleting routers
route: SIOCDELRT: No such process
Adding router 192.168.201.20
```

使用 udhcpc 命令获取 IP

至此已经成功使用英创 ES6210 系列嵌入式 Linux 工控主板连接无线 WiFi，以上步骤均可以通过一个脚本来实现，上电后自动执行脚本，就可以自动连接上设置好的 WiFi，英创公司也已经将编辑好的脚本放在/usr 目录下，用户可以参考这个脚本修改为符合自己应用需求的脚本。

8.2 蓝牙的使用流程

蓝牙（bluetooth）技术是一种低功耗短距离的无线通信技术，被广泛应用于 10 米以内的嵌入式设备通信当中。其最高传输速度根据蓝牙协议的版本不同，有 1Mbps（BR、LE）、2-3Mbps（EDR）、24Mbps（HS）之分。在工业现场，蓝牙技术可以代替串行线缆，实现无线通信。在智能手机普及的今天，通过蓝牙与手机建立连接，手机作为上位机发送指令给下位机，可以实现低成本的 UI 控制方案。

BlueZ 是当前比较成熟的蓝牙协议栈，作为 Linux 系统的官方协议栈，集成在 Linux 内核之中。英创公司在 ES6210 的 Linux 系统中，又移植了 BlueZ 用户空间协议栈和相关工具，使得 ES6210 Linux 平台能够支持蓝牙技术，通过 socket 编程实现蓝牙无线连接，代替串行线缆进行通信。

用户使用蓝牙串口功能主要分为两个步骤：蓝牙功能配置和 socket 应用程序编写。

一、蓝牙功能配置

1、加载 ap2610 蓝牙模块上电驱动

insmod /lib/modules/4.1.14/ap6210_bt_bcm20710.ko

2、加载蓝牙固件，设定波特率、蓝牙地址、使能 hci 等

brcm_patchram_plus --patchram /lib/firmware/ap6210/bcm20702a.hcd --baudrate 3000000 --enable_hci --bd_addr aa:00:55:44:33:22 --no2bytes --tosleep 5000 /dev/ttyS3 1> /dev/null&

3、启动 dbus 后台服务

dbus-daemon --system --nofork --nopidfile &

4、以兼容模式启动 bluetooth 后台服务

/libexec/bluetooth/bluetoothd -C &

5、启动 hci0，并设置 name 和可见属性

hciconfig hci0 up

hciconfig hci0 name es6210

hciconfig hci0 piscan

hciconfig hci0 reset

以上 5 个步骤已经写成一个 shell 脚本 set_bluetooth.sh，用户也可以直接运行该脚本完成以上设置。至此，完成了对蓝牙的设置，可以通过 hciconfig hci0 -a 来查看蓝牙信息，如图 2。这时，其他蓝牙设备就可以搜索到 es6210，点击即可完成配对。

```
[root@EM9280 /mnt/nandflash]#hciconfig hci0 -a
hci0:   Type: BR/EDR  Bus: UART
        BD Address: BB:66:55:44:33:22  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN ISCAN
        RX bytes:1320 acl:0 sco:0 events:76 errors:0
        TX bytes:2286 acl:0 sco:0 commands:76 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
        Name: 'esm9287'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 4.0 (0x6)  Revision: 0x1000
        LMP Version: 4.0 (0x6)  Subversion: 0x220e
        Manufacturer: Broadcom Corporation (15)
```

图 2 使用 hciconfig 查看蓝牙信息

二、Socket 应用编程

蓝牙协议栈中的 RFCOMM 协议实现了对串口 RS232 的仿真，最多能提供两个蓝牙设备之间 60 路的连接。应用程序中，可以使用 socket 进行服务端和客户端的编程，其过程与 TCP/IP 的 socket 通信没有太大区别。

a）环境配置

开发 bluez 协议栈的蓝牙应用需要用到 libbluetooth.so 和相关头文件，需要添加到 eclipse 对应的蓝牙项目中。libbluetooth.so 是编译 bluez 协议栈生产的动态链接库，提供了头文件 bluetooth.h、hci_lib.h、sdp_lib.h 中的函数实体，实现蓝牙地址与常用数据类型的转换、hci 设备和 sdp 服务的一系列操作函数。

1、在项目中新建文件夹 include/bluetooth，其中放入蓝牙协议相关头文件；新建文件夹 lib，其中放动态链接库 libbluetooth.so。

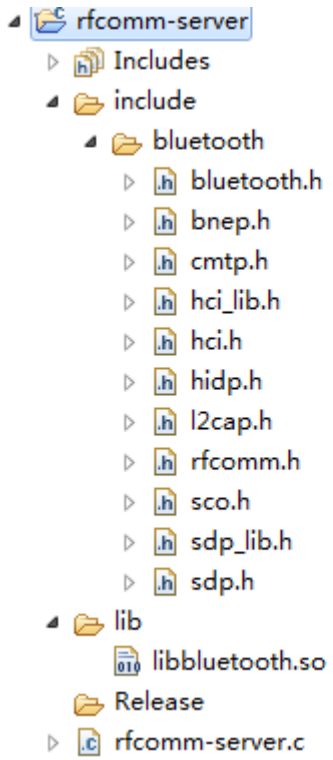


图 4 新建 include 和 lib 文件夹

2、进入项目 Properties 设置，添加项目下的 include 文件夹为 GCC C++ Compiler 和 GCC C Complier 编译器的头文件路径（下图是 GCC C++ Compiler 的设置，GCC C Compiler 设置步骤相同）。

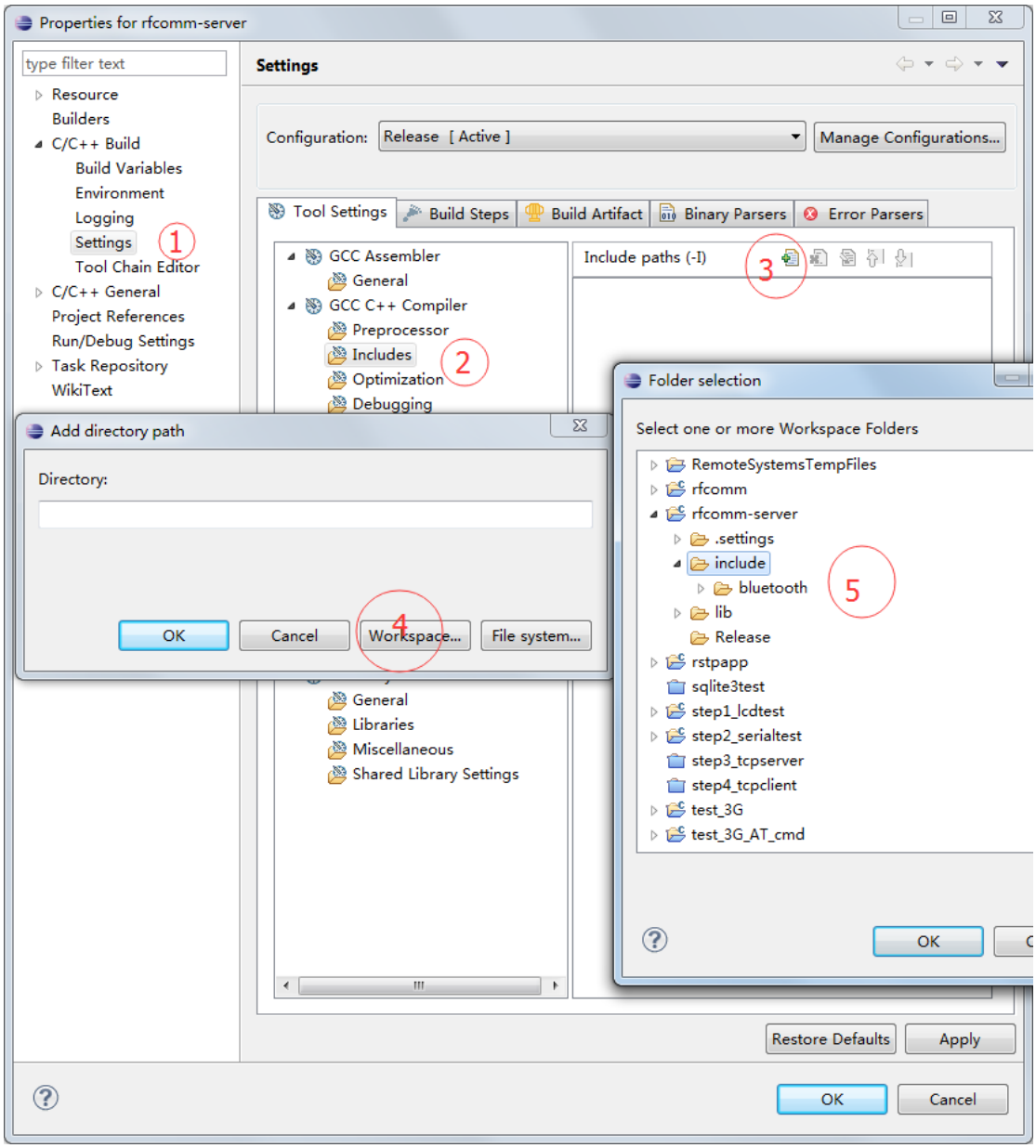


图 5 添加头文件搜索路径

3、为 Sourcery G++ Lite C++ Linker 链接器添加 libbluetooth.so 库文件及搜索路径，如下图。

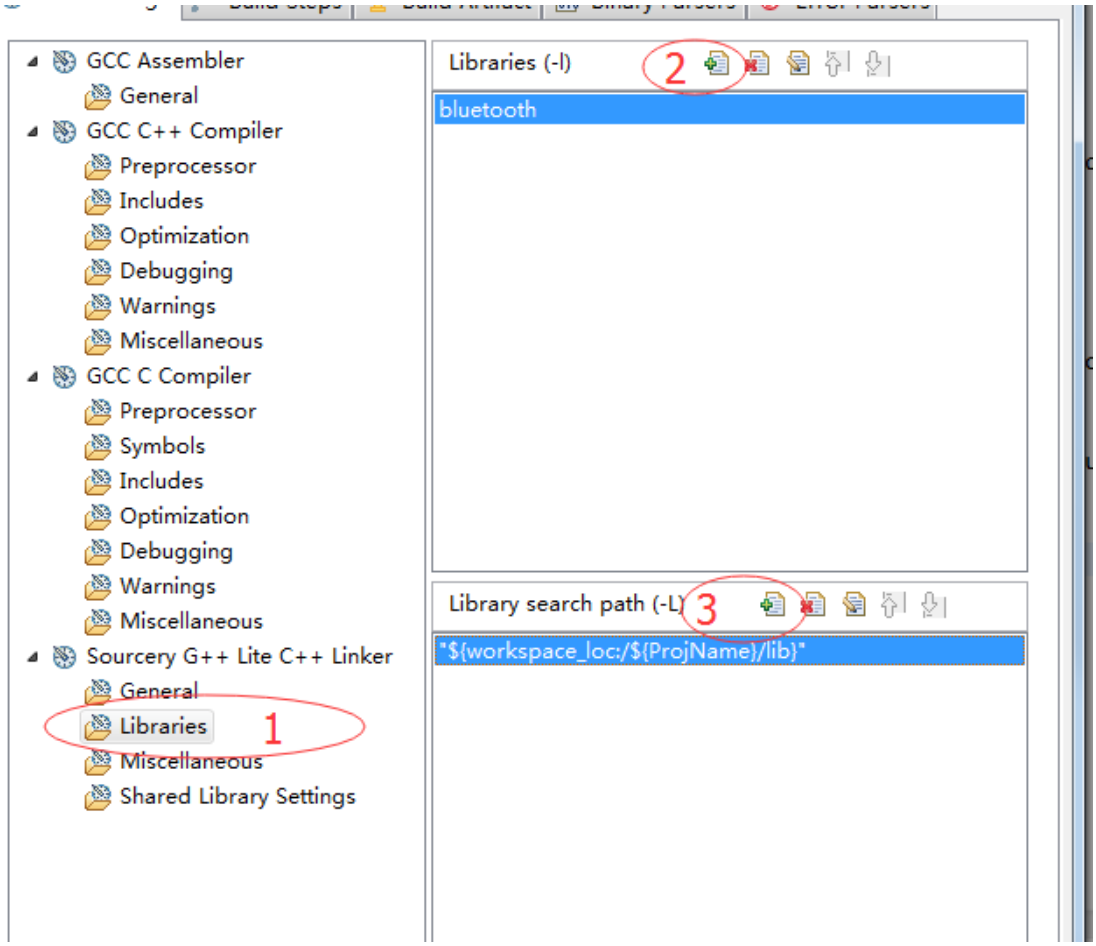


图 6 添加编译库及搜索路径

b）服务端程序

1、申请蓝牙 RFCOMM socket

```
s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
```

2、绑定本地适配器，BDADDR_ANY 默认为第一个可用蓝牙适配器

```
loc_addr.rc_family = AF_BLUETOOTH;
loc_addr.rc_bdaddr = *BDADDR_ANY;
loc_addr.rc_channel = (uint8_t) 1;
bind(s, (struct sockaddr *)&loc_addr, sizeof(loc_addr));
```

3、设置 socket 监听模式，这里只允许建立一个连接

```
listen(s, 1);
```

4、等待连接

```
client = accept(s, (struct sockaddr *)&rem_addr, &opt);
```

5、select 模式读取 socket 数据流

```
while(1)
{
	FD_ZERO(&working_set);
	max_sd = client;
	FD_SET(client, &working_set);
	timeout.tv_sec  = 3 * 60;
	timeout.tv_usec = 0;
	// Call select() and wait 5 minutes for it to complete.
	printf("Waiting on select() %ld sec...\n", timeout.tv_sec);
	int rc_select = select(max_sd + 1, &working_set, NULL, NULL, &timeout);
	// Check to see if the select call failed.
	if (rc_select < 0)
	{
		perror("  select() failed");
		break;
	}
	else if (rc_select > 0)
	{
		if(FD_ISSET(max_sd,&working_set))
		{
			// read data from the client
			bytes_read = read(client, buf, sizeof(buf));
			if( bytes_read > 0 ) {
				printf("received: [%s]\n", buf);
			}
			else
			{
			 break;
			}
			write(client,ack,sizeof(ack));
		}
	}
	// Else if rc_select == 0 then the 5 minute time out expired.
	else
	{
		printf("  select() timed out.\n");
		break;
	}

}
```

6、关闭套接字

```
close(client);

close(s);
```

c）客户端

1、申请蓝牙 RFCOMM socket

```
 s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
```

2、设置蓝牙连接服务器的地址

```
 struct sockaddr_rc addr = { 0 };
// set the connection parameters (who to connect to)
addr.rc_family = AF_BLUETOOTH;
addr.rc_channel = (uint8_t) 1;
str2ba( dest, &addr.rc_bdaddr );
```

3、连接蓝牙服务器

```
// connect to server
status = connect(s, (struct sockaddr *)&addr, sizeof(addr));
```

4、读写 socket 数据流

```
for(i = 0; i < 3; i++)
{
        // send a message
        write(s, message[i], strlen(message[i])+1);
        printf("write \"%s\" to %s\n", message[i],dest);
        bytes_read = read(s, buf, sizeof(buf));
        if( bytes_read > 0 ) {
                printf("received: [%s]\n", buf);
        }

}
```

其中，message[i]为发送内容的地址。

5、关闭 socket

```
 close(s);
```

在一张板子上运行蓝牙 rfcomm 服务程序，在另一张板子上运行蓝牙 rfcomm 客户端程序，如图 7、图 8 所示：

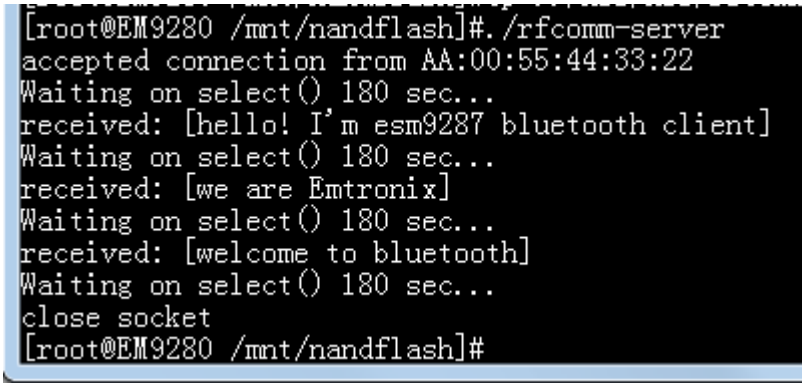


图 7、服务端程序

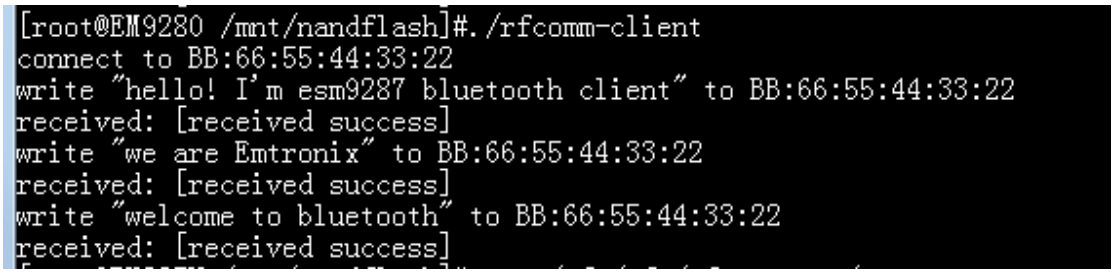


图 8、客户端程序

通过 socket 编程，蓝牙应用程序可以像 tcp/ip 的网络编程一样，建立连接，实现无线通信。同样设置蓝牙的部分可以使用脚本来自动执行，英创公司也将编辑好的一个脚本放在 /usr 目录中供客户参考。

附录 1　版本信息管理表

日期	版本	简要说明
2017 年 06 月	1.0	创建本文档