基于 Mini2440 平台的 RT-Thread RTOS

开发手册

RT-Thread 工作室

2010 年 3 月

1 文档概述

Mini2440 是广州友善之臂计算机科技有限公司出品的一款低价实用的 ARM9 开发板；它采用 Samsung S3C2440 作为微处理器，并采用专业稳定的 CPU 内核电源芯片和复位芯片来保证系统运行时的稳定性。本文档介绍了 RT-Thread 的特点以及如何在 mini2440 开发板上进行 RT-Thread RTOS 的使用和开发。

文档主要从 RT-Thread 的特点，源代码的获取，源代码的目录结构，开发环境的搭建，运行和调试过程以及一些常见问题这几个方面来进行介绍，希望使用 RT-Thread 的开发人员能够少走一些弯路。

2 RT-Thread RTOS 简介

2.1 RT-Thread 概述

实时线程操作系统（RT-Thread）是国内 RT-Thread 工作室精心打造的稳定的开源实时操作系统，历时 4 年的呕心沥血开发，力图突破国内没有小型稳定的开源实时操作系统的局面，它不仅仅是一款开源意义的实时操作系统，也是一款产品级别的实时操作系统，它已经被国内十多所企业所采用，被证明是一款能够稳定持续运行的操作系统。

实时线程操作系统（RT-Thread）不仅是一个单一的实时操作系统内核，它也是一个完整的应用系统，包含了实时、嵌入式系统相关的各个组件：TCP/IP 协议栈，文件系统，图形用户界面 RTGUI，Finsh Shell 等。

实时线程操作系统（RT-Thread）支持的平台如表 2-1 所示：

表 2-1 RT-Thread 支持的硬件平台

芯片	核心	描述	编译环境
AT91SAM7X256	ARM7TDMI	基本系统，TCP/IP 协议栈	GNU GCC 及 RealView MDK
LPC2478	ARM7TDMI	基本系统，文件系统，TCP/IP 协议栈	GNU GCC 及 RealView MDK
S3C2440	ARM920T	基本系统，面向 0.4.x 分支	GNU GCC 及 RealView MDK
STM32F103VB/ZE	ARM Cortex M3	基本系统，TCP/IP 协议栈，文件系统，图形界面(仅 STM32F103ZE 支持)	GNU GCC 及 RealView MDK
LM3S	ARM Cortex M3	基本系统，TCP/IP 协议栈，文件系统	GNU GCC 及 RealView MDK
I386	X86	基本系统（运行于真实机器或 QEMU 模拟器）	Linux，GNU GCC

在开发过程中一些其他移植，例如 ATMEL AT91SAM6S，AT91RM9200，XScale 270，ColdFire 等，此处没一一列出。

2.2 内核特点

RT-Thread 是基于面向对象的方式开发的实时内核，标准内核体积为 9K Byte 左右，在主频为 180M 的 arm9 芯片上运行，线程切换时间和中断切换时间分别在 7us 和

6us 左右，因为其小型的特点也可以看成是一个嵌入式操作系统。主要有如下特点：
(1) 面向对象方式的实时核心（但依然保留了 C 语言的优雅、小巧风格）；
(2) 默认 32 线程优先级的全抢占式实时内核（亦可配置成 256 线程优先级）；相同优先级线程时间片轮转调度；
(3) 相同优先级线程实施时间片可配置的分时时间片轮转调度；
(4) 线程间同步机制：信号量和防止优先级翻转的互斥锁；
(5) 完善高效的线程间通信机制，包括邮箱，消息队列和事件；
(6) 支持线程挂起和唤醒的固定内存块管理及线程安全的动态内存堆管理；
(7) 向上层提供基于名字的统一接口设备驱动模型；
(8) 支持各个内核对象(如信号量对象，消息队列对象)的裁剪；
(9) 支持可嵌套的中断；
(10) 支持线程堆栈溢出检测；

2.3 网络组件 LwIP

LwIP 是瑞士计算机科学院（Swedish Institute of Computer Science）的 Adam Dunkels 等开发的一套用于嵌入式系统的开放源代码 TCP/IP 协议栈，它在包含完整的 TCP 协议实现基础上实现了小型的资源占用，因此它十分适合于使用到嵌入式设备中，占用的体积大概在几十 kB RAM 和 40KB ROM 代码左右。
由于 LwIP 出色的小巧实现，而功能也相对完善（包含相对完整的 BSD 风格 socket 编程），用户群比较广泛。实时线程操作系统（RT-Thread）采用 LwIP 做为默认的 TCP/IP 协议栈，同时根据小型设备的特点对其进行再优化，体积相对进一步减小。

2.4 文件系统组件

RT-Thread/DFS 是 RT-Thread 的文件系统组件，它有点类似 Linux 的虚拟文件系统，但是更加小巧，通过它能够适配不同的文件系统格式，如个人电脑上常用的 Fat 文件系统格式和嵌入式系统常用的 Flash 文件系统格式。当前 RT-Thread 0.3.0 中加入了支持 Fat 格式的 FatFS 和 EFSL 开源文件系统，由于许可证的原因，会逐渐过渡到使用 FatFS。RT-Thread/DFS 有以下特点。
(1) 提供 Posix 文件系统接口。
(2) 支持多分区挂载。
(3) 支持 FAT12/16/32 文件系统格式。

2.5 图形界面组件

RT-Thread/GUI (简称 RTGUI)是 RT-Thread 的图形界面组件，它是和 RT-Thread 相配套的组件，目前的许可证限制其不得移植到 RT-Thread 之外的其他操作系统(代码中

可能有指向其他许可证，但请添加上面的补充条款)。

RT-Thread/GUI 是专为 RT-Thread 操作系统开发的，并在相应的地方采用了 RT-Thread 特有的功能，它是一款几乎从零代码开始编写的，拥有独立知识产权的图形界面（不包括 JPEG、PNG 等图形库部分）。在 RT-Thread 许可证的条款下，能够免费的在商业产品中使用。RT-Thread/GUI 的特性包括：

(1) 多线程图形用户界面；

(2) 依赖于 RT-Thread 线程调度器的实时图形用户界面；

(3) 创新式的在嵌入式系统中引入面板的概念，缩小了多线程，多窗口图形用户界面编程代价：

1) workbench，对应多线程；

2) view，对应不同的用户界面视图；

3) window，对应多窗口；

(4) C 语言方式的全面向对象设计：

1) 对象具备运行时类型信息；

2) 对象自动销毁，使得内存的管理更为轻松；

(5) 丰富的控件支持：

1) button，checkbox，radiobox

2) textbox

3) progressbar，slider

4) listview，filelist_view

详细介绍 RTGUI 的文档见《RTGUI 编程指南》。

2.6 Finsh Shell 组件

Finsh Shell 组件提供了一套供用户在命令行操作的接口,主要用于调试，查看系统信息，主要有以下功能。

(1) 获取系统运行时信息，如各种 RT-Thread 内核对象的动态信息

(2) 能够对任意寄存器和内存地址进行读写操作

(3) 能够对任意寄存器和内存地址进行读写操作

在 9.2.2 一节“使用 Finsh shell 调试”中详细介绍了这些功能。

3 RT-Thread 在 mini2440 上的开发

3.1 开发状况

RT-Thread 的发展方向是依赖于其中的 rtconfig.h 配置文件，对各种组件做剪裁，形成一套细粒度、全功能的非常稳定的实时操作系统，开发方向主要有两个分支，其中一个面向资源及其有限的嵌入式系统，如 ARM Cortex M3 系列芯片平台，另一个方向是面向资源丰富的系统，如 ARM7/9 系列芯片平台。在 mini2440 平台上的开发就是属于这个分支。开发方式采用开源协作的方式，开发人员主要通过 Gtalk, MSN, QQ 群沟通，通过 SVN 提交代码。

目前 mini2440 分支已经完成 RT-Thread 及其所有组件的支持。

(1) 完成串口驱动，SD 卡驱动，dm9000 网卡驱动，LCD 驱动，触摸屏驱动。

(2) 支持 SD 卡的 Fat 文件系统。

(3) 支持 LwIP TCP/IP 协议栈及一些网络应用，如 Ftp server, Goahead webserver, tftp server/client 等。

(4) 支持 RTGUI。

当前正在开展的工作是进一步优化和完善 RT-Thread 及其组件，增加 USB 功能以及考虑基于 mini2440 平台和 RT-Thread 展开进一步的开源项目，如开源人机界面项目。同时，RT-Thread 0.4.0 版本的演进也会基于 mini2440 平台来开发，到 0.4.0 版本，动态模块加载功能会加入进来，RT-Thread 的内核会得到进一步的增强，动态特性也会更好。但是不管怎样，稳定性始终是 RT-Thread 追求的首要目标。

3.2 开发截图

为了更加直观的表达 RT-Thread 在友善之臂 mini2440 平台上的进展，下面展示了 RT-Thread & RTGUI 在 mini2440 上的部分截图。

图 3-1 RTGUI 的 window 控件

图 3-1 为 RTGUI 的 window 控件 Demo 图。

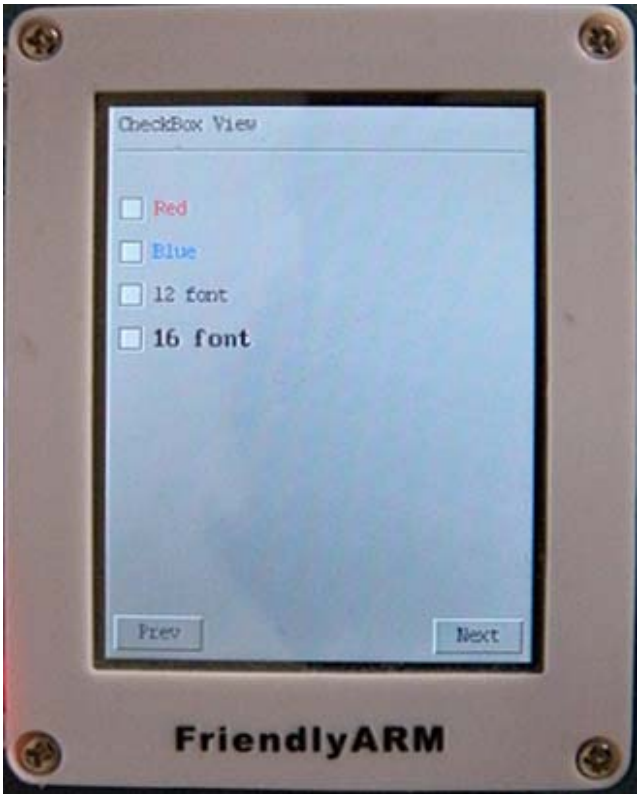

图 3-2 RTGUI 的 checkbox 控件

图 3-2 为 RTGUI 的 Check Box 控件的 Demo 图。

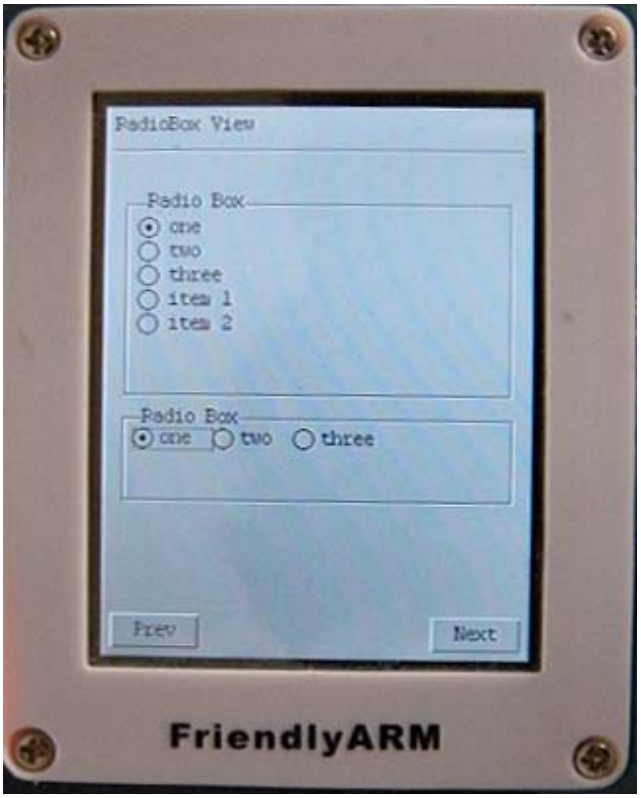

图 3-3 RTGUI 的 RadioBox 控件

图 3-3 为 RTGUI 的 RadioBox 控件的 Demo 图。

图 3-4 RTGUI 的 TextBox 控件

图 3-4 为 RTGUI 的 TextBox 控件的 Demo 图。

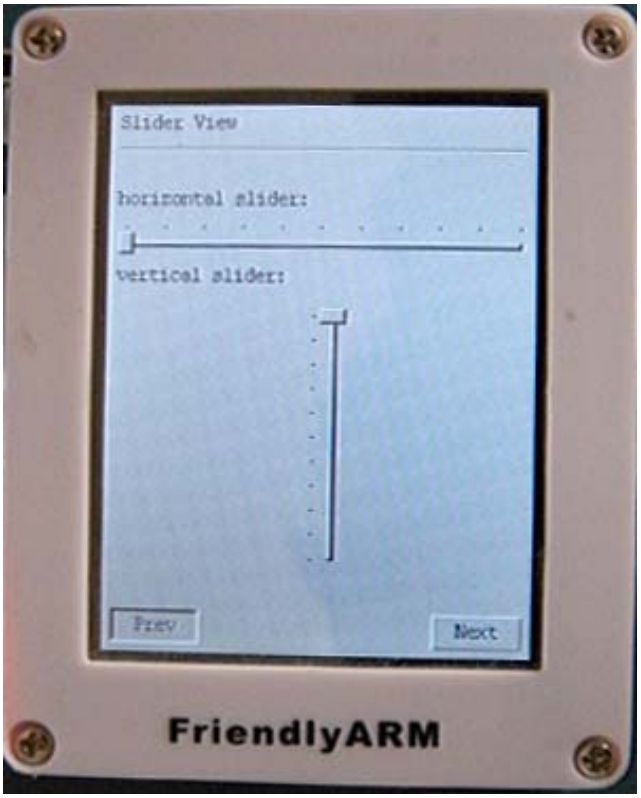

图 3-5 RTGUI 的 Slider 控件

图 3-5 为 RTGUI 的 Slider 控件的 Demo 图。

图 3-6 RTGUI 的 window 控件

图 3-6 RTGUI 的 window 控件为 RTGUI 的 list view 的 Demo 图。

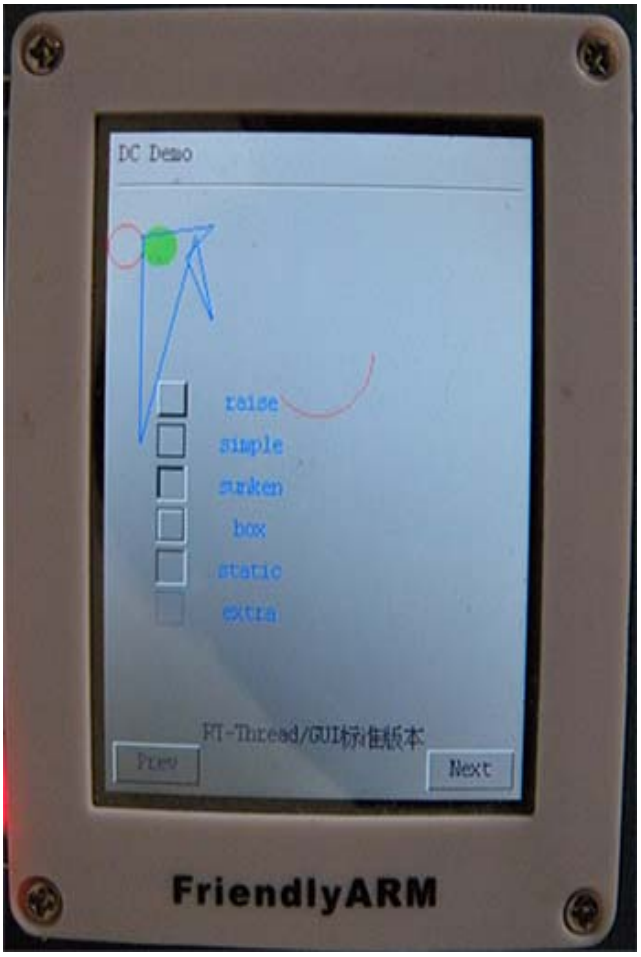

图 3-7 RTGUI 的绘图控件

图 3-7 RTGUI 的绘图控件为 RTGUI 的 DC Demo 图。

4 获取 RT-Thread 源码

4.1 SVN 源码管理

RT-Thread 作为一款开源实时操作系统，它的内核源代码和所有组件均是开放，免费的。RT-Thread 的代码寄托于 Google SVN 版本服务器，并且用于每个开发人员的同步化开发。对于一般的用户，可以通过匿名方式获得 RT-Thread 的最新代码，一般来说需要懂些 SVN 的知识，或者如果是在 Windows 上，可以装那个俗称小乌龟的 Tortoise SVN 客户端。为一切简单化，RT-Thread 开发团队提供了另外一个非常好的方法。以下两小节详细介绍这两种方法。

4.2 使用 TortoiseSVN 下载

使用 TortoiseSVN 前请先安装该软件。安装完毕后在开始菜单，程序项中会出现如图 4-1 所示的 TortoiseSVN 程序图标。

图 4-1 TortoiseSVN 程序图标

同时在鼠标右键菜单中会多出 SVN Checkout 和 TortoiseSVN 两项菜单。开始下载 RT-Thread 时，首先在本地磁盘建立文件夹 RT-Thread，然后在该文件夹图表上点击右键并在右键菜单中选择 SVN Checkout 项。如图 4-2 所示。

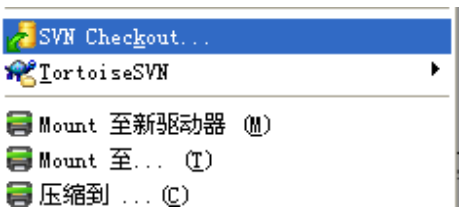

图 4-2 安装 Tortoise SVN 后的右键菜单项

此时会弹出如图 4-3 所示的 Checkout 对话框，在 URL of repository 输入框中输入网址 http://rt-thread.googlecode.com/svn/trunk，点击 OK。

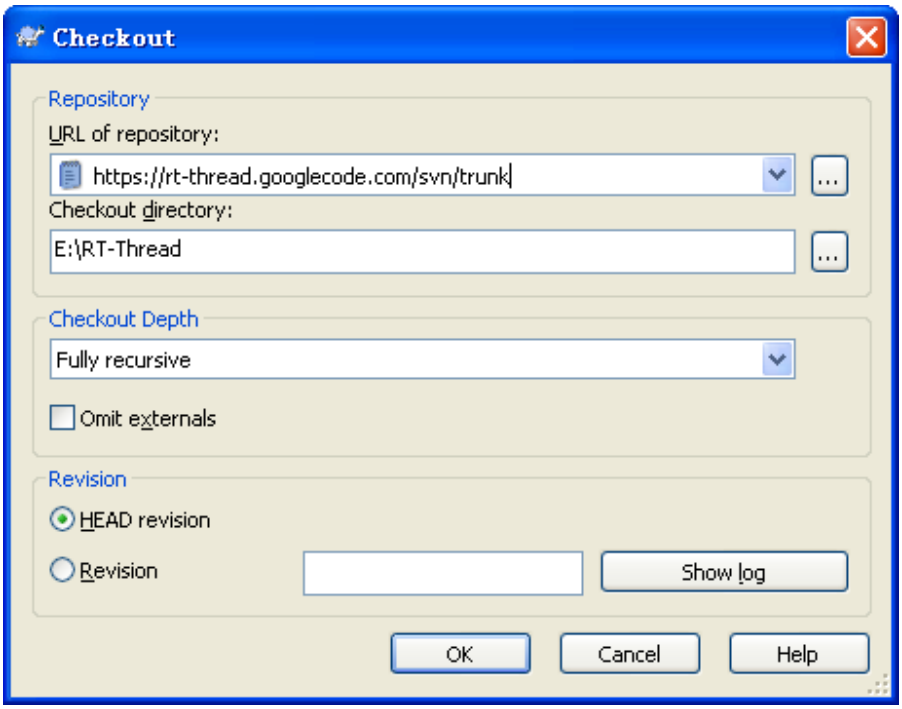

图 4-3 Tortoise SVN 的 Checkout 对话框

这时就进入下载界面，开始更新代码，如图 4-4 所示。

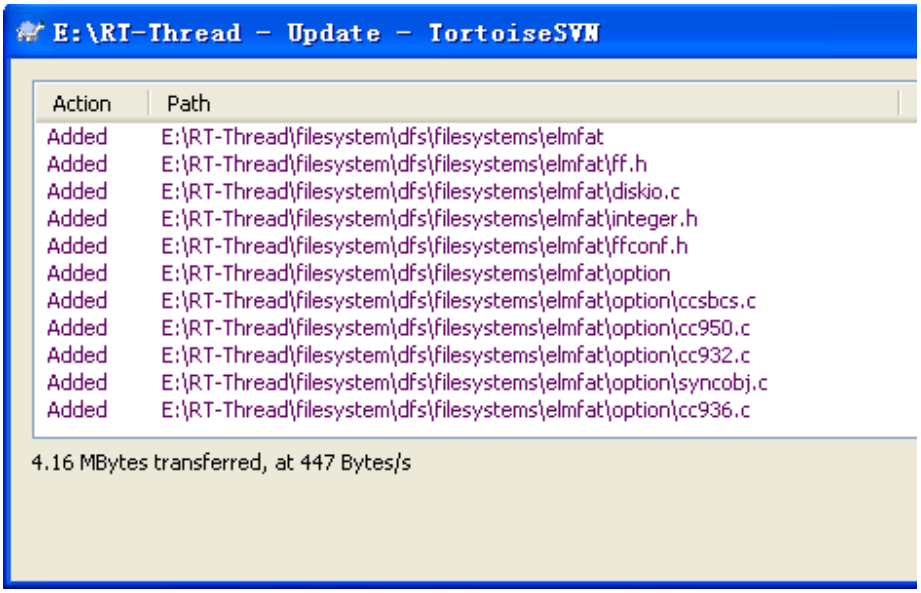

图 4-4 Tortoise SVN 的 Update 界面

直至所有 RT-Thread 代码下载到本地文件夹。如图 4-5 所示。

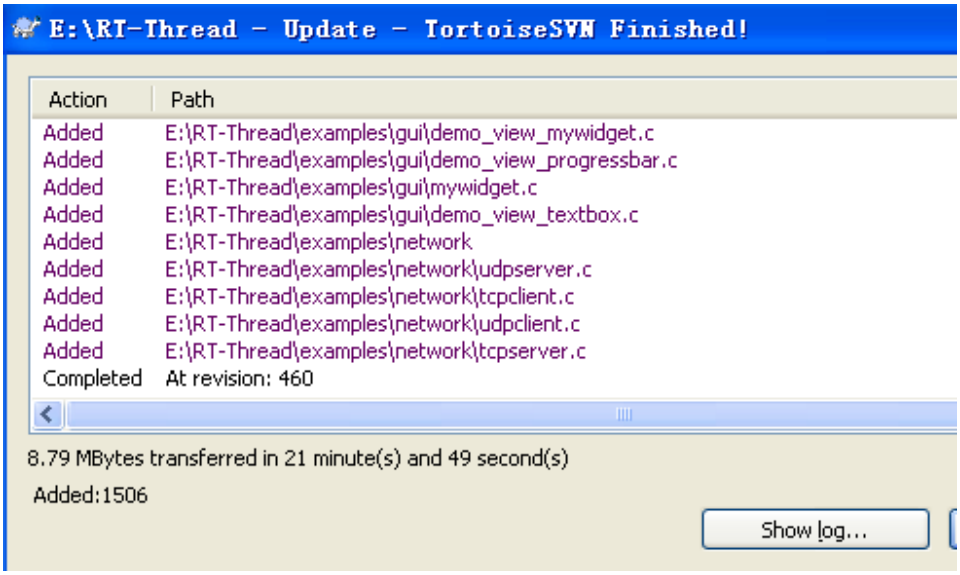

图 4-5 Tortoise SVN 更新完毕

4.3 使用 TortoiseSVN 提交代码

开发者提交代码，首先需要通过邮件或者论坛站内信件方式向 RT-Thread 工作室提供您的 gmail 帐号，这样就可以给你开通代码提交权限了。同时，你可以从 Google code 上获取你的代码提交密码，具体方式如下：
首先用你的 gmail 帐号登录 Google code，登录后在网页右上角会有 Profile 的链接，如图 4-6 所示。

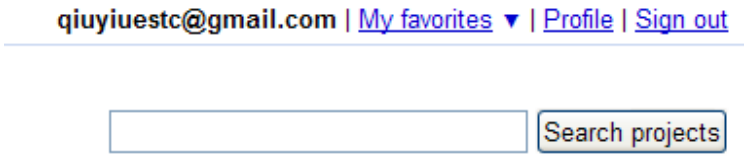

图 4-6 Google code 登录界面

点击 Profile 后出现如下图所示的界面，在 Settings 的 Tab 中就可以看到你的代码提交密码了，如图 4-7 所示。

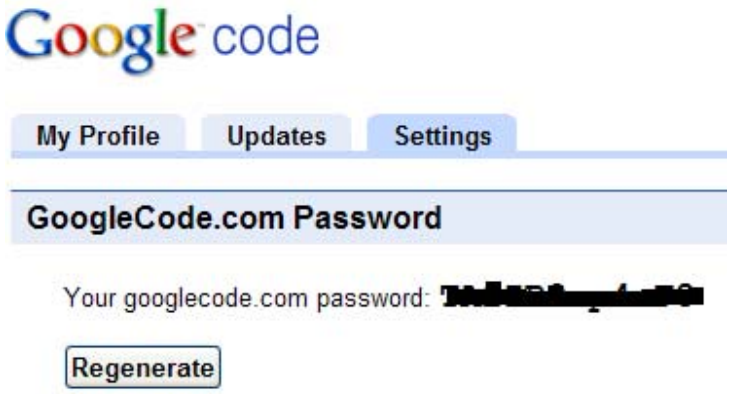

图 4-7 Google code 页面帐号信息

有一点需要特别注意的是提交代码时的路径是
https://rt-thread.googlecode.com/svn/trunk/
而普通用户匿名检出代码的路径是
http://rt-thread.googlecode.com/svn/trunk/
两者差异在于前者是 https，而后者是 http。
前面准备工作完毕，然后就可以在需要提交的文件或文件夹图标上点击右键，选择 SVN Commit，如图 4-8 所示。

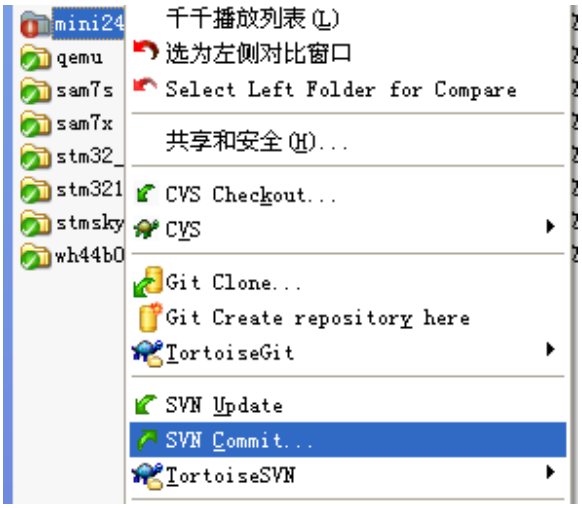

图 4-8 通过 SVN Commit 提交代码

选择完毕以后会出现如图 4-9 所示的提交界面，在输入框中敲入提交信息，点击 OK，就开始向服务器提交代码了。

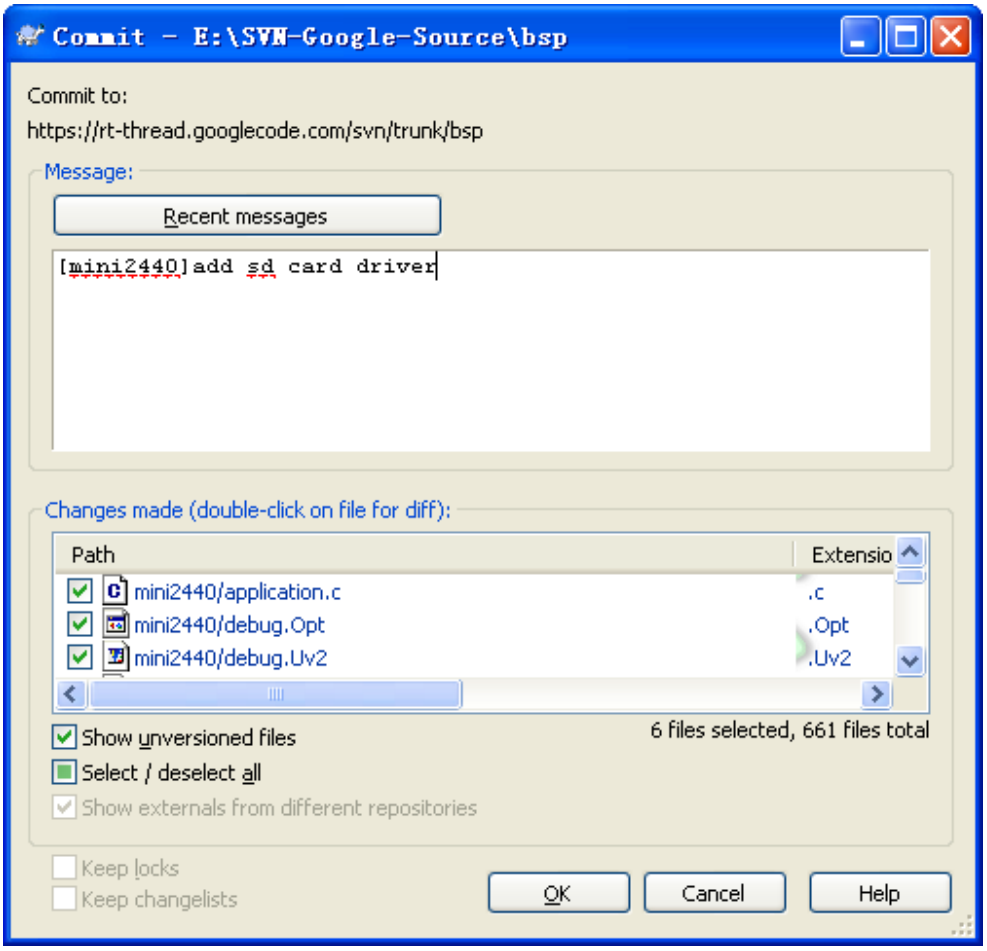

图 4-9 Tortoise SVN 的提交界面

4.4 快速下载代码

这儿介绍另外一种下载 RT-Thread 源代码的方法。首先从以下地址下载 RT-Thread 开发人员制作的下载工具：
http://www.rt-thread.org/phpbb/download/file.php?id=295
下载完，解压到您选择的本地目录，然后运行目录中的 get-update-rtt.bat 批处理文件，就开始从服务器下载代码，如图 4-10 所示，最终在该目录下会创建 rt-thread 目录，其中包含了 RT-Thread 所有源代码。

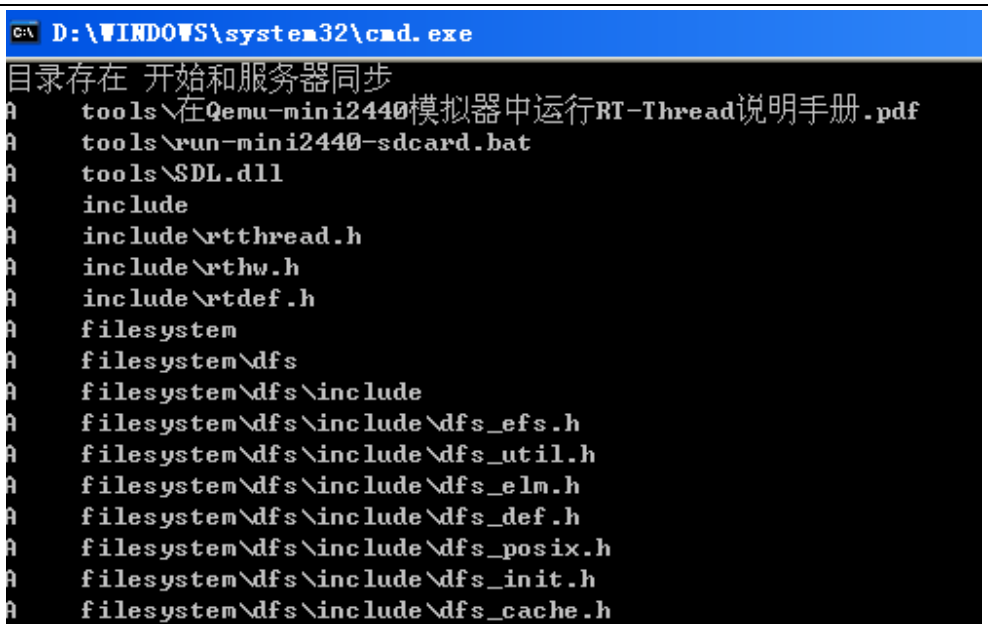

图 4-10 使用命令行方式下载

需要做代码同步时，也只需运行该批处理文件。

5 RT-Thread 源代码目录结构

5.1 顶层目录

图 5-1 描述了 RT-Thread 根目录下所有的子目录和文件。

图 5-1 RT-Thread 根目录

表 5-1 分别对根目录下所有文件进行介绍。

表 5-1 RT-Thread 根目录下所有文件的描述

目录	描述
Bsp	板级支持包
Examples	例子和测试程序
Filesystem	设备文件系统组件
Finsh	Finsh shell 组件
Include	RT-Thread 对外开放的头文件
Libc	RT-Thread 需要用到的小型 libc 库
Libcpu	芯片体系架构相关的文件
Net	TCP/IP 网络协议栈组件
Rtgui	图形界面接口 RTGUI 组件
Src	RT-Thread 内核
Tools	运行 RT-Thread 的模拟器工具
Authors	作者信息
Copying	版权信息

5.2 bsp 目录

图 5-2 bsp 目录

图 5-2 描述的是 bsp 目录，其中存放的是 RT-Thread 的板级支持包文件，每种开发板都对应一个子目录，从这个子目录可以看出目前 RT-Thread 支持的开发板类型。在移植 RT-Thread 到其他开发板上时，需要在该 bsp 目录下创建相应的子目录。

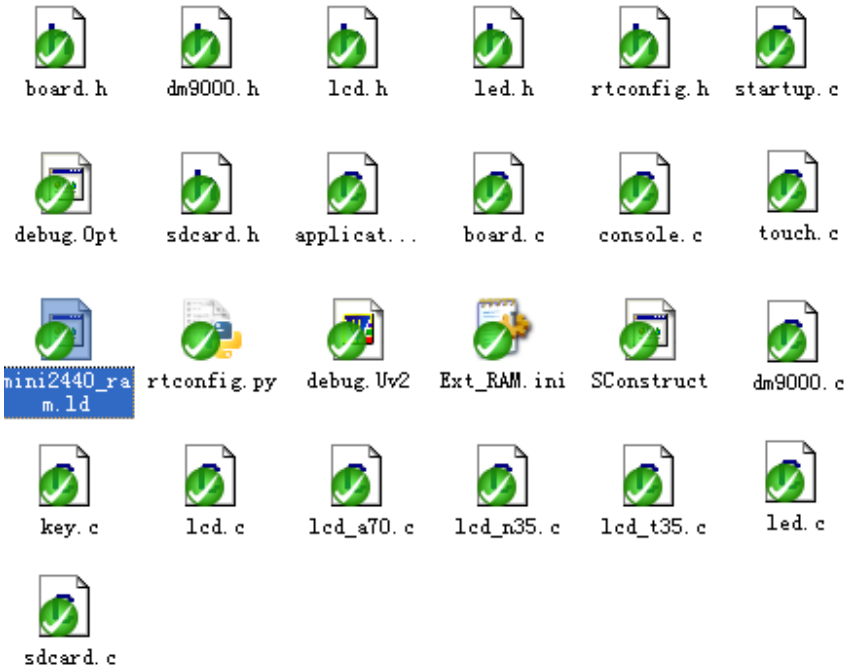

图 5-3 bsp/mini2440 子目录中的文件

图 5-3 为 bsp/mini2440 子目录中的文件，board.c 为与开发板相关的初始化文件，startup.c 为系统启动相关文件， application.c 为启动应用线程的文件，其他如 dm9000.c，touch.c 等都是驱动文件；rtconfig.py 和 SConstruct 是编译相关文件，用来控制编译，debug.Uv2 为 MDK 空工程，用来调试 RT-Thread。

5.3 examples 目录

图 5-4 examples 目录

图 5-4 描述的是 Examples 目录，其中存放的是 RT-Thread 内核和组件相关的例程，用作测试和应用示例。目前有内核，文件系统，网络，RTGUI 的例程源码，随着开发的深入，该目录下的文件会持续增加。

5.4 libcpu 目录

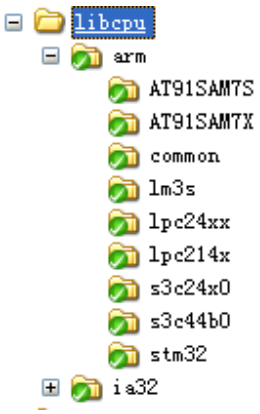

图 5-5 libcpu 目录

图 5-5 描述的是 libcpu 目录，其中存放的是与芯片体系架构相关的文件，目前 RT-Thread 主要支持 ARM 和 IA32 两种体系架构，而 ARM 体系架构下又细分了众

多的 SOC。RT-Thread 支持的体系架构和 SOC 可以从该目录体现出来。

5.5 filesystem 目录

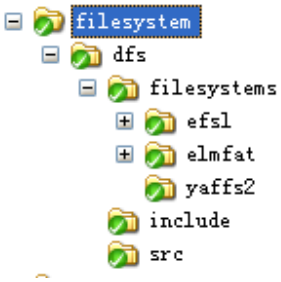

图 5-6 fileysystem 目录

图 5-6 描述的是 fileystem 目录，其中存放的是与 RT-Thread/DFS 文件系统组件相关的文件，filesystem/dfs 子目录下有 filesystems 和 src，include 三个子目录，filesystmes 子目录下存放的是与具体文件系统格式实现相关的目录，如支持 fat 文件系统格式的 efsl 和 elmfat，支持 flash 文件系统格式的 yaffs2。src 和 include 子目录下存放的是 DFS 虚拟文件系统实现相关的文件。

5.6 net 目录

net 目录下有 lwip 和 app 两个子目录，其中 lwip 目录存放的是开源 TCP/IP 协议栈。app 目录下存放的是一组面向网络的应用程序，很多应用程序都采用 BSD socket 来作为网络接口。目前 app 目录下包括： ping , tcpecho server, udpecho server, ftp server, tftp client, sntp client, chargen server。

此外，RT-Thread 也移植了较为强大的 Goahead Webserver，因为许可证的原因，没有将它放入 SVN 中管理，在 RT-Thread 论坛上能够下载到移植完毕的 Goahead Webserver。

6 搭建 RT-Thread 开发环境

6.1 scons 命令行构建系统

对于大的工程来说，使用 IDE 来管理总是有很多限制，例如大多数 IDE 不能够支持在工程中存在相同文件名的文件（即使是路径不相同）。而且用 IDE 来做 bulid，速度会慢一些。
RT-Thread 0.2.x 以前都是采用的 GNU Make + GNU GCC 的方式来编译，但 Makefile 的组织太过复杂，特别是要做编译选项，会很麻烦（编译选项不能够在多个 Makefile 之间自由传递）。 GNU 还有一种方式，名为 automake, autoconfig，不过当自动产生的 Makefile 比自己的代码还多的多的时，有时就不得不思考，程序员是在写代码，还是 automake&autoconfig 在写程序员。
RT-Thread 发展到 0.3.x，主要的编译器切换到了 RealView ARM Compiler，所以默认没有发布出带 Makefile 的版本。
scons 是构建在 python 脚本的基础上的，功能很多。依赖于 python，能够在 scons 的脚本中做很多的事，控制也灵活。例如：scons 本身是携带分析代码的功能的，最具意义的是它分析 C/C++头文件预处理的功能。通过这个预处理分析，能够获知已经定义和未定义的宏。推广到 RT-Thread 的 rtconfig.h 配置文件，就能够通过 scons 的分析功能来分析 rtconfig.h 文件，从而自动获得 RT-Thread 的组件使能情况。
构建 scons 环境所需工具有：
pywin32-212.win32-py2.5 , scons-1.2.0.d20090113.win32
以下是工具下载地址
http://www.python.org/ftp/python/2.5.2/python-2.5.2.msi
http://sourceforge.net/projects/scons/files/scons/1.2.0.d20091224/
RT-Thread 这边的镜像：
http://www.rt-thread.org/download/python-2.5.2.msi
http://www.rt-thread.org/download/scons-1.2.0.d20091224.win32.exe
下载完毕后安装这两个工具，先安装 python，再安装 scons，安装完后，请打开 dos 命令行窗口，执行 set PATH=c:\Python2.5\Scripts;%PATH%。
这样就配置完毕了 scons 编译构建系统。此构建系统支持所有命令行方式编译环境，如 MDK 的 armcc 编译环境，GNU 的 gcc 编译环境。

6.2 MDK 编译环境

MDK 编译环境需要安装 RealView MDK(正式版或评估版) 3.5+，因为会用到它的编

译链接器和调试器。可以从 Keil 官方网站下载 RealView MDK 评估版，MDK 运行界面如图 6-1 所示。

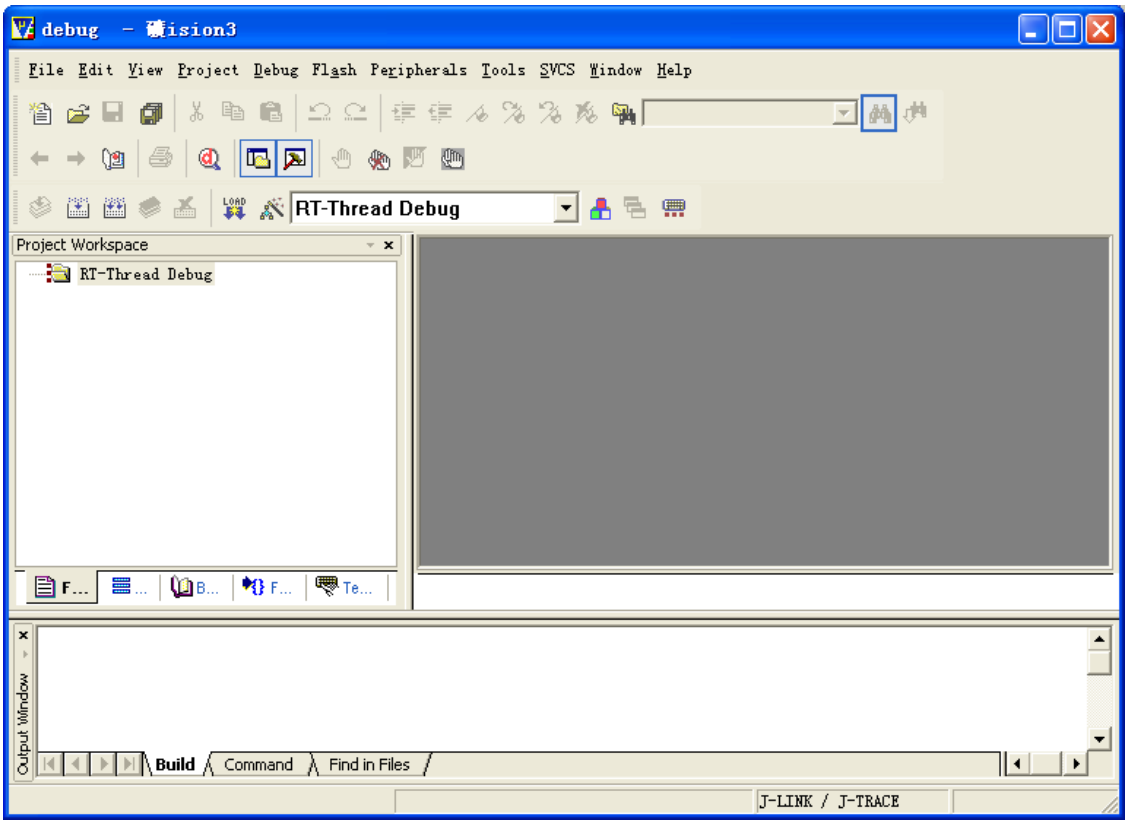

图 6-1 Keil MDK 运行界面

6.3 GNU GCC 编译环境

对于习惯于使用 GNU GCC 编译环境的开发人员，可以不必使用 MDK 编译环境，请自行下载安装 CodeSourcery Sourcery_G++，安装完毕即搭建好 GNU GCC 编译环境。

6.4 开始编译之旅

编译环境搭建好以后，就可以开始编译之旅了，首先打开 RT-Thread \bsp\mini2440\rtconfig.py 文件，找到如图 6-2 所示代码

```
CROSS_TOOL      = 'keil'

if  CROSS_TOOL == 'gcc':
        PLATFORM        = 'gcc'
        EXEC_PATH       = 'E:/Program Files/CodeSourcery/Sourcery G++ Lite/bin'
elif CROSS_TOOL == 'keil':
        PLATFORM        = 'armcc'
        EXEC_PATH       = 'E:/Keil'
BUILD = 'debug'
```

图 6-2 rtconfig.py 配置编译环境代码片段

配置 CROSS_TOOL 为 MDK 或者 GNU 编译环境，如果是 MDK 编译环境，设置 CROSS_TOOL = ‘keil’，如果是 GNU 编译环境，设置 CROSS_TOOL = ‘gcc’，并且将 EXEC_PATH 设置成你工具所在的路径。
如果你是 MDK 3.4 或者以下版本，请找到 EXEC_PATH += ‘/arm/bin40/’这一行，将它改为 EXEC_PATH += ‘/arm/bin31/’
配置完毕后，打开 cmd 窗口，进入 RT-Thread/bsp/mini2440 目录，输入 scons，回车，便开始进行编译过程，开始编译过程如图 6-3 所示。

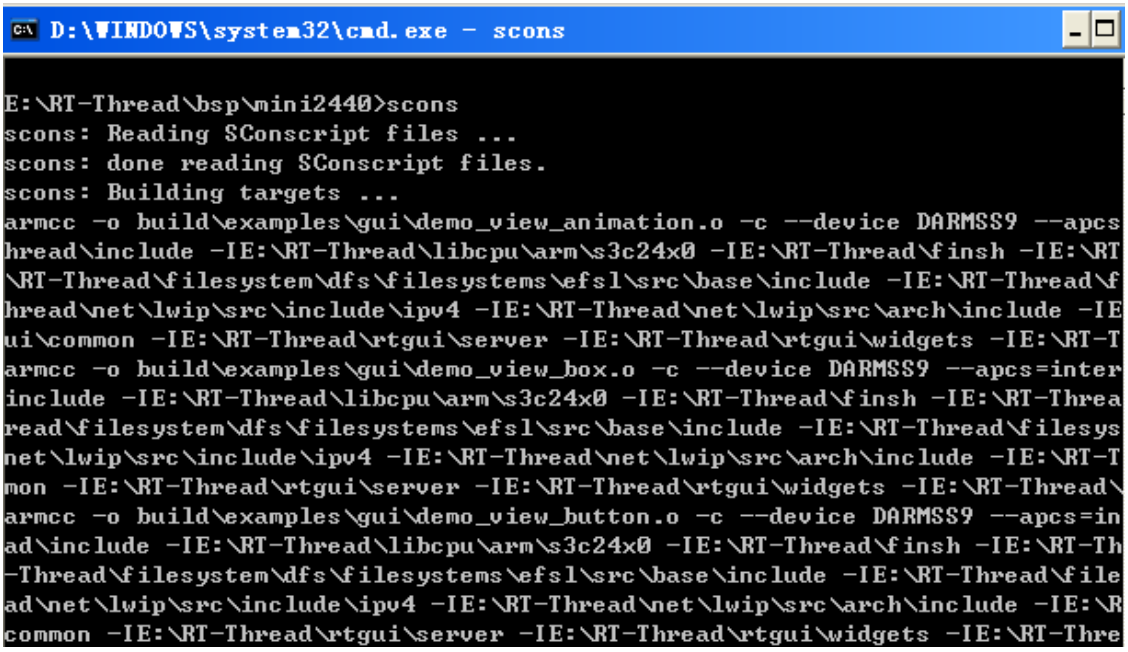

图 6-3 开始编译

直至最后链接完毕，生成 axf 和 bin 文件，如图 6-4 所示。

图 6-4 编译，链接后生成的 bin 和 axf 文件

7 mini2440 开发板配置

不同的友善之臂 mini2440 开发板主要区别在于 LCD 显示屏不同，因此需要对不同的 LCD 屏进行配置。在 RT-Thread 的 bsp 目录下已经提供了三种 LCD 屏的驱动，它们分别是 A70，N35 和 T35，需要根据 LCD 屏类型来修改 RT-Thread/bsp/mini2440/rtconfig.py 中的配置。首先在配置文件中找到如下图 7-1 所示的代码，修改成与开发板对应的 LCD 屏选项。

```
# panel options
# 'PNL_A70','PNL_N35', 'PNL_T35'
RT_USING_LCD_TYPE = 'PNL_T35'
```

图 7-1 LCD 屏配置选择代码片段

8 RT-Thread 组件裁剪和配置

8.1 Finsh shell 组件的配置

8.1.1 Finsh shell 组件的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，找到如图 8-1 所示代码。

```
069 /* SECTION: FinSH shell options */
070 /* Using FinSH as Shell*/
071 #define RT_USING_FINSH
072 /* Using symbol table */
073 #define FINSH_USING_SYMTAB
074 #define FINSH_USING_DESCRIPTION
```

图 8-1 使用 Finsh shell 组件

使用 Finsh shell 组件功能时，将宏 RT_USING_FINSH 打开，如图 8-1 所示。

```
069 /* SECTION: FinSH shell options */
070 /* Using FinSH as Shell*/
071 /* #define RT_USING_FINSH */
072 /* Using symbol table */
073 /* #define FINSH_USING_SYMTAB */
074 /* #define FINSH_USING_DESCRIPTION */
```

图 8-2 关闭 Finsh shell 组件

关闭功能，将宏 RT_USING_FINSH 注释掉，如图 8-2 所示。默认打开。

8.2 TCP/IP 网络协议栈组件的配置

TCP/IP 网络协议栈需要实现的是网卡驱动，在 RT-Thread/mini2440 分支中已经提供 dm9000 驱动。

8.2.1 网络功能的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，如图 8-3 所示。

```
093 /* SECTION: lwip, a lighwight TCP/IP protocol stack */
094 /* Using lighweight TCP/IP protocol stack*/
095 #define RT_USING_LWIP
```

图 8-3 配置 LwIP 的代码片段

使用网络模块功能时，将宏 RT_USING_LWIP 打开。关闭网络功能，将该宏注释掉，即/* define RT_USING_LWIP */。默认打开。

8.2.2 DHCP 功能的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，如图 8-4 所示。

```
123 /* Using DHCP*/
124 /* #define RT_LWIP_DHCP */
```

图 8-4 配置 DHCP 功能的代码片段

使用 DHCP 功能时，将宏 RT_LWIP_DHCP 打开。关闭网络功能，将该宏注释掉，即/* define RT_LWIP_DHCP */。默认关闭。

8.2.3 SNMP 功能的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，如图图 8-5 所示。

```
00131: /* Enable SNMP protocol */
00132: /* #define RT_LWIP_SNMP */
00133:
```

图 8-5 配置 SNMP 功能的代码片段

使用 SNMP 功能时，将宏 RT_LWIP_SNMP 打开。关闭网络功能，将该宏注释掉，即/* define RT_LWIP_SNMP */。默认关闭。

8.2.4 网络 IP 地址的配置

网络地址的配置包括 IP 地址，网关地址，和子网掩码的配置，可以选择两种配置方

式，一种是静态配置方式。配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件。

```
128 /* ip address of target*/
129 #define RT_LWIP_IPADDR0 192
130 #define RT_LWIP_IPADDR1 168
131 #define RT_LWIP_IPADDR2 0
132 #define RT_LWIP_IPADDR3 30
```

图 8-6 配置 IP 地址的代码片段

如图 8-6 所示，默认 IP 地址为 192.168.0.30，需要在这里改成您需要设置的 IP 地址。

```
134 /* gateway address of target*/
135 #define RT_LWIP_GWADDR0 192
136 #define RT_LWIP_GWADDR1 168
137 #define RT_LWIP_GWADDR2 0
138 #define RT_LWIP_GWADDR3 1
```

图 8-7 配置网关的代码片段

如图 8-7 所示，默认网关地址为 192.168.0.1，需要在这里改成您需要设置的网关地址。

```
140 /* mask address of target*/
141 #define RT_LWIP_MSKADDR0    255
142 #define RT_LWIP_MSKADDR1    255
143 #define RT_LWIP_MSKADDR2    255
144 #define RT_LWIP_MSKADDR3    0
```

图 8-8 配置子网掩码的代码片段

如图 8-8 所示，默认子网掩码为 192.168.0.30，需要在这里改成您需要设置的子网掩码。

第二种配置方式是在使用 Finsh Shell 的前提下，在系统运行时动态修改 IP 地址，网关地址，子网掩码。如图 8-9 所示。

```
 \ | /
- RT - Thread Operating System
 / | \ 0.3.0 build Jan  3 2010
 2006 - 2009 Copyright by rt-thread team
part[0], begin: 122368, size: 477.392MB
finsh>>File System initialized!
TCP/IP initialized!

finsh>>set_if("192.168.0.30","192.168.0.1","255.255.255.0")
        536885396, 0x20003894
finsh>>
```

图 8-9 使用 Finsh Shell 修改网络地址

8.2.5 影响网速的几个参数配置

实际调试中发现，在 LwIP 协议栈中，对网络速度影响最大的参数主要是 MSS，TCP_WND 和 TCP_SND_BUF。
MSS 就是 TCP 数据包每次能够传输的最大数据分段。为了达到最佳的传输效能 TCP 协议在建立连接的时候通常要协商双方的 MSS 值，这个值 TCP 协议在实现的时候往往用 MTU 值代替（需要减去 IP 数据包包头的大小 20Bytes 和 TCP 数据段的包头 20Bytes）所以往往 MSS 为 1460。通讯双方会根据双方提供的 MSS 值得最小值确定为这次连接的最大 MSS 值。
TCP_WND 与 TCP 接收相关，能够做多个数据包接收，然后只发一个 ACK 确认。
TCP_SND_BUF 与发送相关。
这几个参数可以在 lwip_opts.h 文件中配置
在 RT-Thread/mini2440 分支，这几个参数默认配置为 MSS = 1024，TCP_WND = 1024，TCP_SND_BUF = 1024*10。

8.3 文件系统组件的配置

RT-Thead/mini2440 分支目前主要支持 SD 卡上的 FAT 格式文件系统

8.3.1 文件系统组件的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，如图 8-11 所示。

```
084 #define RT_USING_DFS
085 /* SECTION: DFS options */
```

图 8-10 配置文件系统功能的代码片段

使用 DFS 文件系统组件时，将宏 RT_USING_DFS 打开。关闭文件系统组件，将该宏注释掉，即/* define RT_USING_DFS */。默认打开。

8.4 RTGUI 组件的配置

8.4.1 RTGUI 组件的使用与关闭

配置步骤：打开 RT-Thread/bsp/mini2440/rtconfig.h 文件，如图 8-11 所示。

```
00178: /* SECTION: RTGUI support */
00179: /* using RTGUI support */
00180: #define RT_USING_RTGUI
00181:
```

图 8-11 RTGUI 的配置代码片段

使用 RTGUI 图形用户界面组件时，将宏 RT_USING_RTGUI 打开。关闭 RTGUI 图形用户界面功能，将该宏注释掉，即/* define RT_USING_RTGUI */。默认打开。

9 运行和调试 RT-Thread

9.1 运行 RT-Thread

9.1.1 利用开发板 BIOS 加载运行 RT-Thread

(1) 连接好 mini2440 开发板电源，串口线，USB 线，并设置开发板为 Nor Flash 启动系统，打开 DNW，上电启动开发板，如图 9-1 所示。
(2) 保证 USB 驱动已经安装好。

图 9-1 DNW 界面

(3) 点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000，如图 9-2 所示。

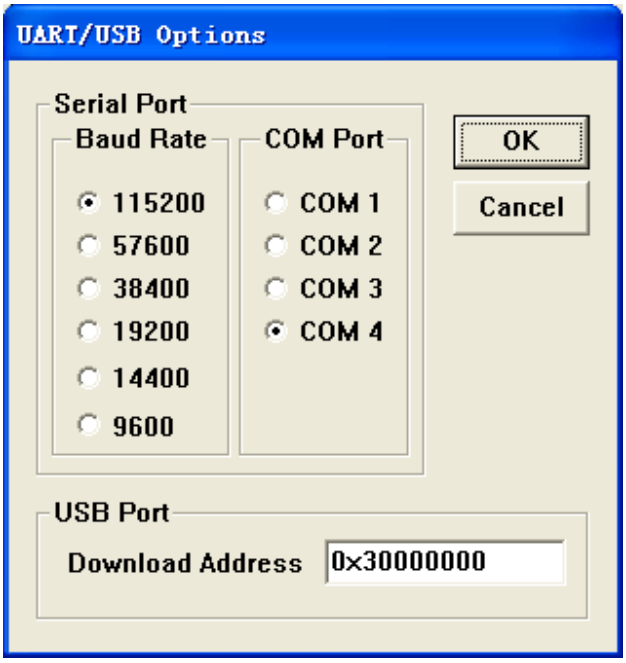

图 9-2 设置 USB 下载地址

(4) 用 DNW 连接开发板串口，这时在 DNW 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息，如图 9-3 所示。

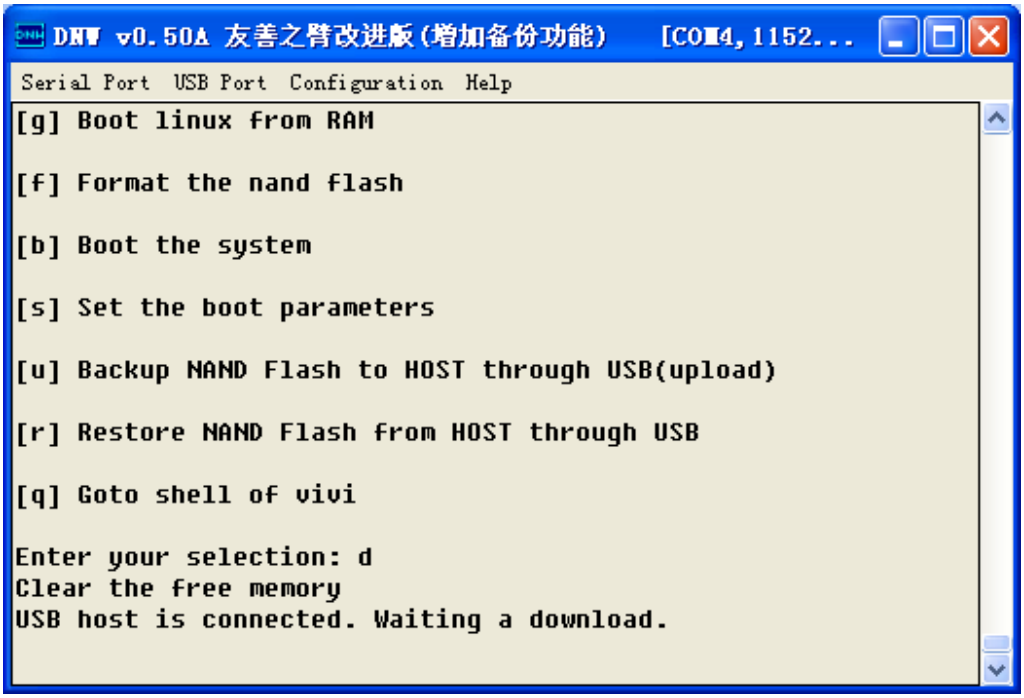

图 9-3 USB 下载等待提示

(5) 点击 DNW 程序的“USB Port”->“Transmit”，选择编译出的 rt-thread.bin 映象文件，如图 9-4 所示。

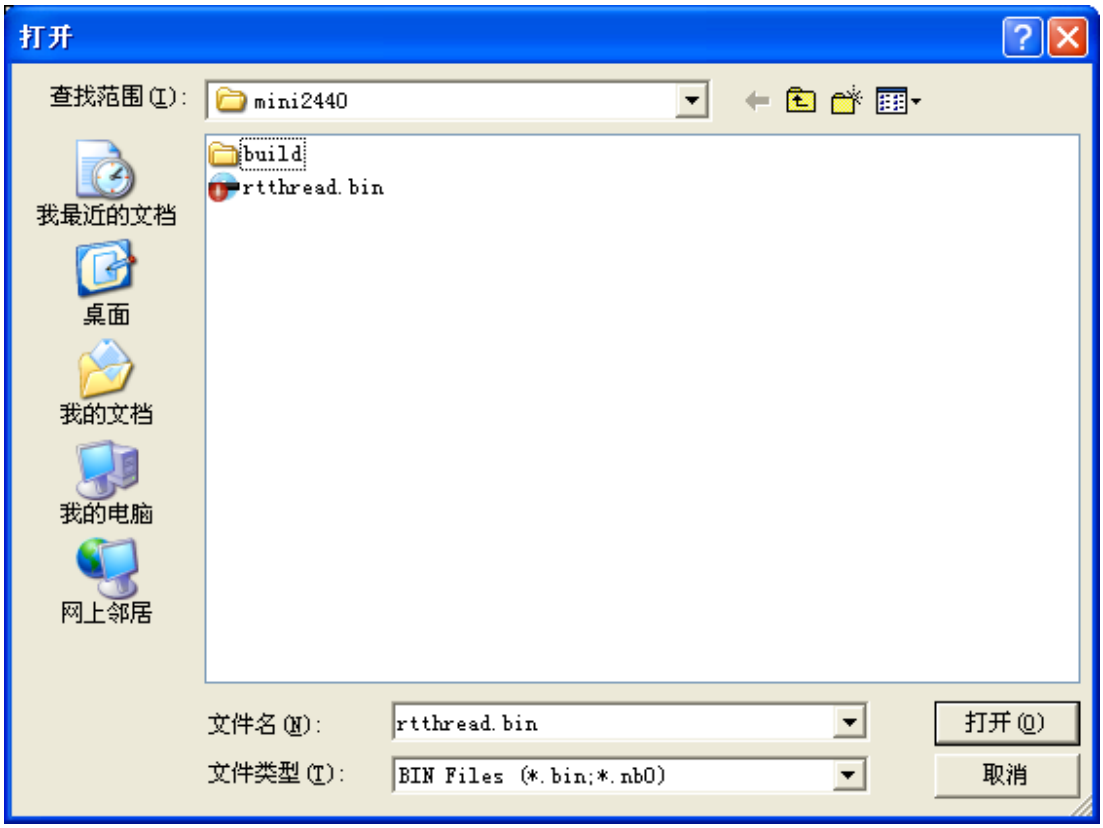

图 9-4 选择烧写到开发板的文件

(6) 下载运行后，RT-Thread 就自动运行了，同时 LCD 上出现 RTGUI 界面，串口输出信息如图 9-5 所示。

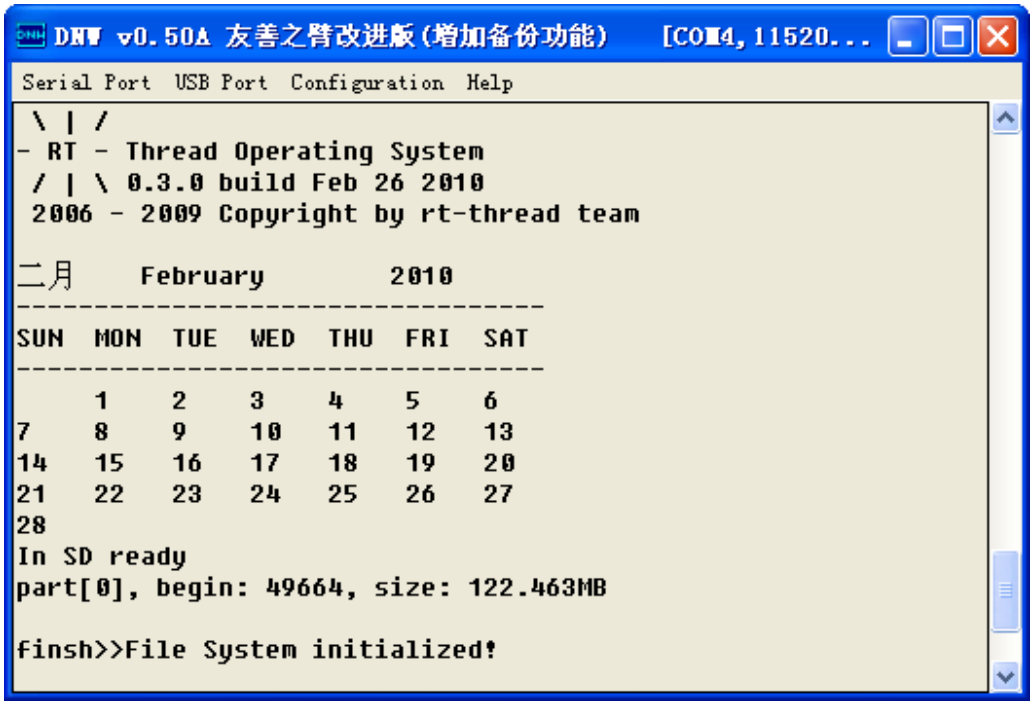

图 9-5 RT-Thread 运行时串口输出信息

9.1.2 使用 MDK+JLink 加载运行 RT-Thread

使用 MDK+JLink 加载运行 RT-Thread 的方式和调试 RT-Thread 的方式是一致的，因此这部分与下一章节“调试 RT-Thread”一起讲述。

9.2 调试 RT-Thread

9.2.1 用 MDK+JLink 调试 RT-Thread

打开 RT-Thread/bsp/mini2440/ debug.Uv2 工程文件，如图 9-6 所示。

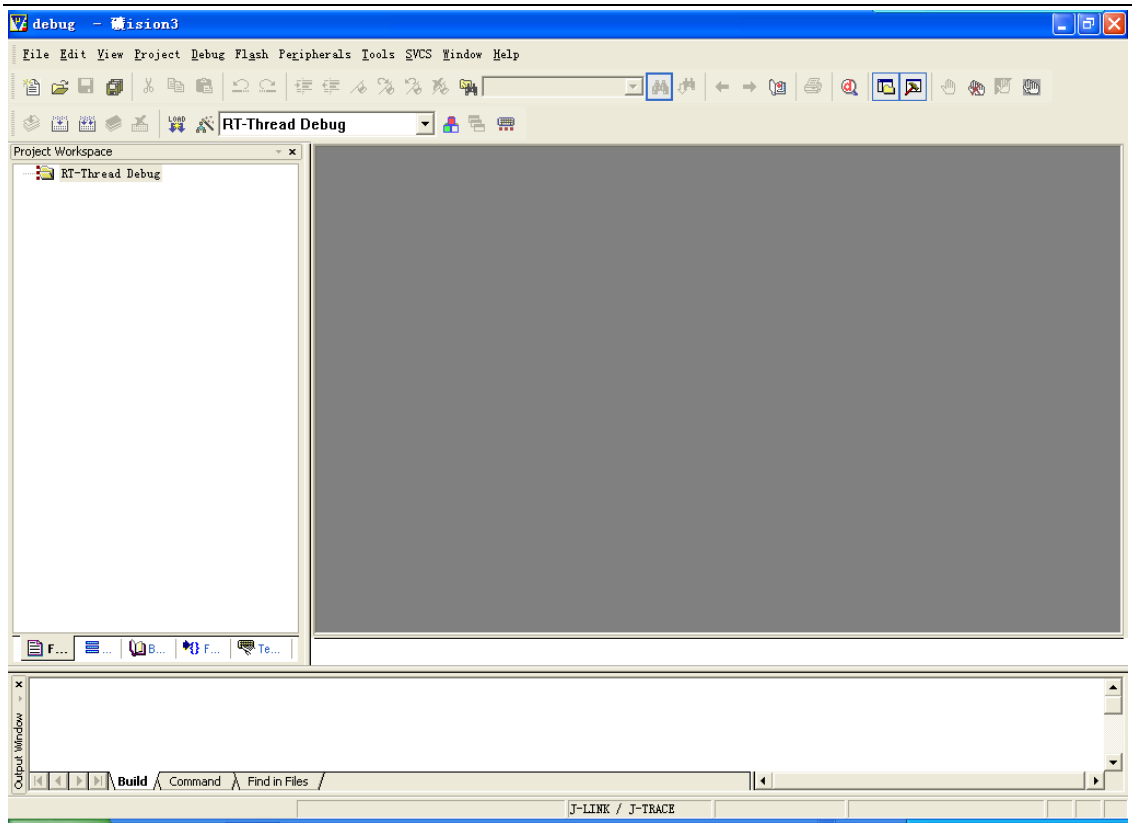

图 9-6 MDK 工程

点击“Debug”->“Start/Stop Debug Session”，可以看到 MDK IDE 左下方的进度条一直在前进，如图 9-7 所示，此过程为 JLink 将 RT-Thread 的 axf 文件下载到内存中。

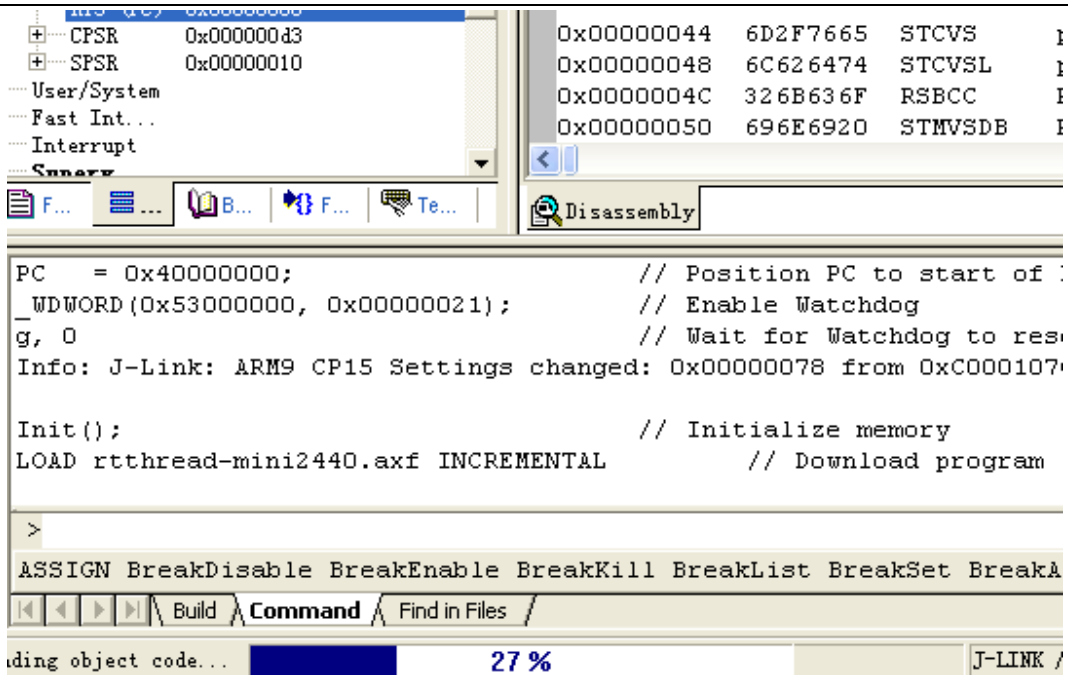

图 9-7 下载目标文件到开发板

点击“File”->“Open”打开 RT-Thread 目录下需要调试的文件，设置断点，如图 9-8 所示。

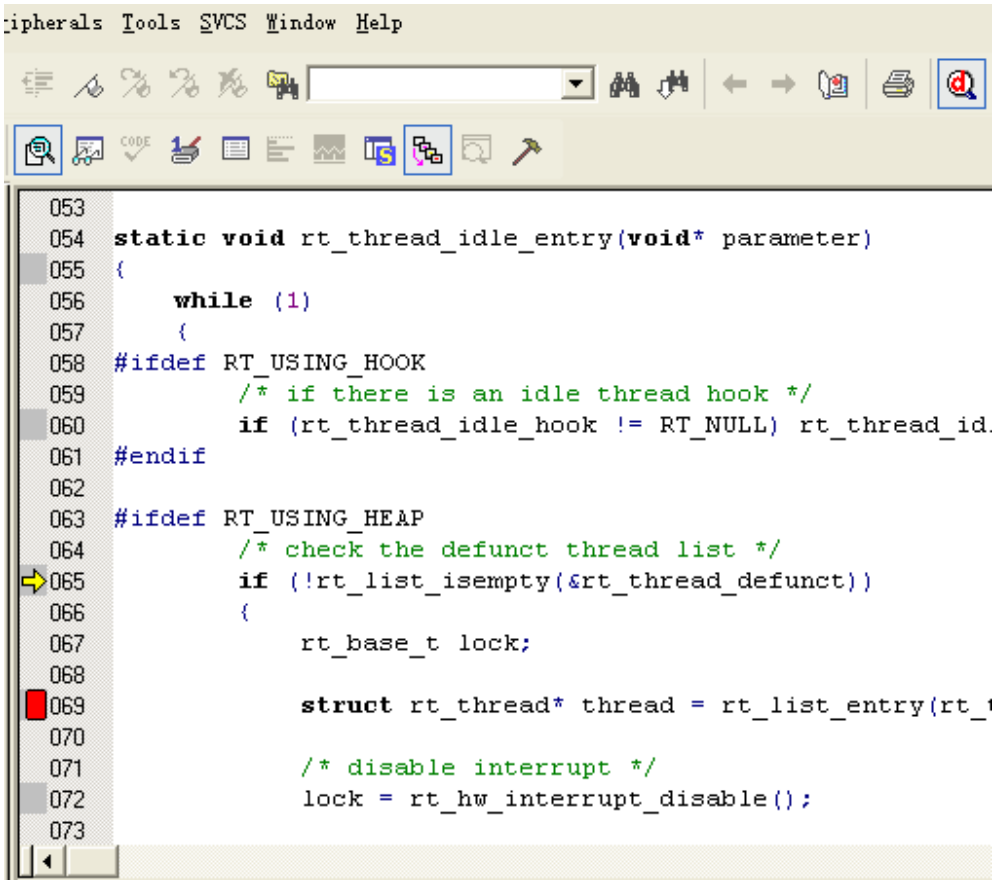

图 9-8 在 MDK IDE 中设置断点

(1) 点击“Debug” -> “Run”，程序就运行起来了，直到设置断点的地方。此时，威力强劲的 MDK 调试工具就任你驾驭了。

9.2.2 使用 Finsh shell 调试

Finsh 是 RT-Thread 的 shell 组件，表面看来方便，易用，内部实则非常复杂和强大。它包含了一个可接受 C 语言表达式的 shell 系统，而且被实现成类似于一个编译器，实现了纯手工编写的完整词法分析，语法分析，然后产生中间二进制代码，最后放到虚拟机中运行。对于用户而言，主要有以下功能：

(1) 获取系统运行时信息，如各种 RT-Thread 内核对象的动态信息。

```
finsh>>list()
--Function List:
set_date          -- set date(year, month, day)
set_time          -- set time(hour, minute, second)
list_date         -- list date
hello             -- say hello world
version           -- show RT-Thread version information
list_thread       -- list thread
list_sem          -- list semaphone in system
list_event        -- list event in system
list_mutex        -- list mutex in system
list_mailbox      -- list mail box in system
list_msgqueue     -- list message queue in system
list_mempool      -- list memory pool in system
list_timer        -- list timer in system
list_device       -- list device in system
list              -- list all symbol in system
ls                -- list directory contents
_mkdir            -- make a directory
rm                -- remove files or directories
cat               -- print file
set_if            -- set network interface address
```

图 9-9 list()命令的输出

RT-Thread 运行起来后，它会在串口输出 RT-Thread 的信息以及 Finsh>>的命令提示符，以等待用户输入命令。首先输入 list()，回车，就会显示出所有系统命令，如图 9-9 所示。下面的内容详细介绍部分系统命令，表 9-1 为部分系统命令的描述。

表 9-1 Finsh shell 内核基本命令及其描述

命令	描述
Version	列表当前使用 RT-Thread 的版本信息
List_thread	列表所有线程信息
List_sem	列表所有信号量信息
List_event	列表所有事件信息
List_mutex	列表所有互斥量信息
List_mailbox	列表所有邮箱信息
List_msgqueue	列表所有消息队列信息
List_mempool	列表所有内存池信息
List_timer	列表所有定时器信息
List_device	列表所有设备信息
List	列表所有系统命令信息

```
finsh>>list_thread()
 thread   pri  status      sp     stack size max used   left tick  error
-------- ---- ------- ---------- ---------- ---------- ---------- ---
test     0x19 ready   0x0000004c 0x00000200 0x0000004c 0x00000008 000
tcpip    0x80 ready   0x00000044 0x00001000 0x00000044 0x00000014 000
etx      0x90 ready   0x00000044 0x00000400 0x00000044 0x00000010 000
erx      0x90 ready   0x00000044 0x00000400 0x00000044 0x00000010 000
wb       0x19 ready   0x000001dc 0x00000800 0x000003ec 0x00000007 000
rtgui    0x0f suspend 0x000001bc 0x00000800 0x000001dc 0x00000004 000
tidle    0xff ready   0x00000044 0x00000100 0x00000044 0x00000020 000
tshell   0x14 ready   0x0000024c 0x00000800 0x000003b8 0x0000004c 000
led      0xc8 ready   0x00000044 0x00000200 0x00000044 0x00000014 000
init     0x50 init    0x00000074 0x00000800 0x00000328 0x00000011 000
       0, 0x0000
```

图 9-10 list_thread()命令的输出

图 9-10 所示为输入 list_thread()命令后，RT-Thread 向串口输出的实际运行时线程信息。从以上信息可以看出当前系统中有多少线程，每个线程的名称，优先级，运行状态，堆栈地址，堆栈大小，最大使用堆栈，剩余运行 tick 时间，错误信息。

```
finsh>>list_sem()
semaphore v   suspend thread
--------  --- --------------
sem1      001 0
sem0      001 0
e0        000 0
eth_lock  001 0
tx_ack    001 0
topwin    001 0
rtgui     001 0
uart      000 0
sem_sd1   001 0
sem_sd0   001 0
timer     1631 0
       0, 0x0000
```

图 9-11 list_sem()命令的输出

图 9-11 所示为输入 list_sem()命令后，RT-Thread 向串口输出的实际运行时所有信号量的信息。

```
finsh>>list_mutex()
mutex    owner    hold suspend thread
-------- -------- ---- --------------
dlock    (NULL)   0000 0
       0, 0x0000
```

图 9-12 list_mutex()命令的输出

图 9-12 所示为输入 list_mutex()命令后，RT-Thread 向串口输出的实际运行时所有互斥量的信息。

```
finsh>>list_event()
event    set        suspend thread
-------- ---------- --------------
        0, 0x0000
```

图 9-13 list_event()命令的输出

图 9-13 所示为输入 list_event()命令后，RT-Thread 向串口输出的实际运行时所有事件的信息。

```
finsh>>list_mailbox()
mailbox  entry size suspend thread
-------- ----  ---- --------------
mbox0    0000  0008 0
etxmb    0000  0032 0
erxmb    0000  0032 0
        0, 0x0000
```

图 9-14 list_mailbox()命令的输出

图 9-14 所示为输入 list_mailbox()命令后，RT-Thread 向串口输出的实际运行时所有邮箱的信息。

```
finsh>>list_msgqueue()
msgqueue entry suspend thread
-------- ----  --------------
workbenc 0004  0
rtgui    0000  1:rtgui
        0, 0x0000
```

图 9-15 list_msgqueue()命令的输出

图 9-15 所示为输入 list_msgqueue()命令后，RT-Thread 向串口输出的实际运行时所有消息队列的信息。

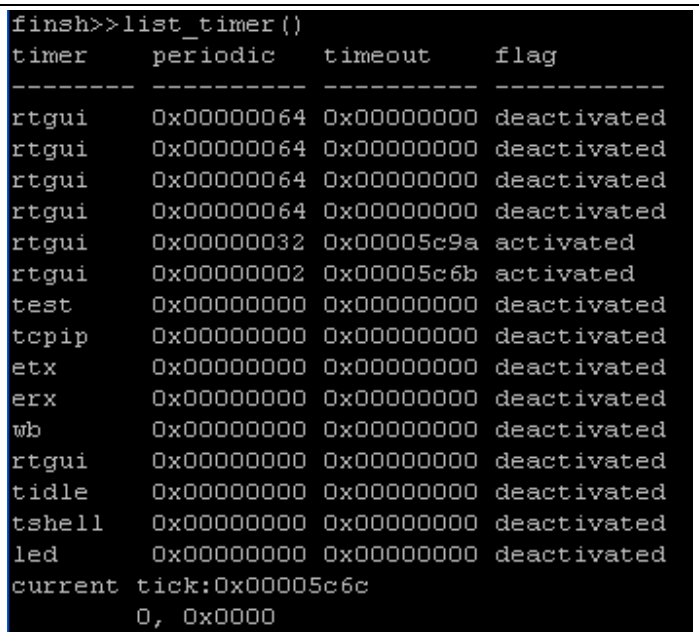

图 9-16 list_tmer()命令的输出

图 9-16 所示为输入 list_timer()命令后，RT-Thread 向串口输出的实际运行时所有定时器的信息。

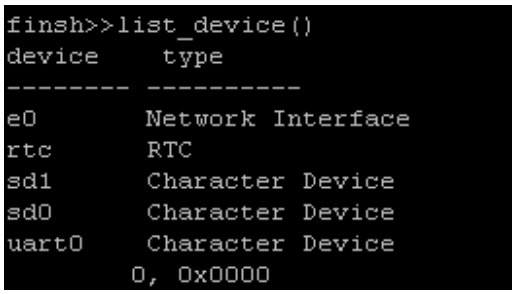

图 9-17 list_device()命令的输出

图 9-17 所示为输入 list_device()命令后，RT-Thread 向串口输出的实际运行时所有设备的信息。

(2) 能够对任意寄存器和内存地址进行读写操作

FinSH 是一类非常特殊的 shell 系统，即不类似于 DOS 的命令行用法，也不类似于 UNIX 类的 sh 用法。而是一种看起来陌生而实际上非常非常熟悉的 C 语言模式：在命令行状态下，一条命令相对于 C 中的一条语句。例如下面的语句：

```
unsigned int    *memory_addr;          /* 定义变量 memory_addr 类型 */
memory_addr = 0x30020000;              /* memory_addr 指向地址 0x30020000 */
*memory_addr = 0x12345678;             /* 向地址 0x30020000 赋值 */
```

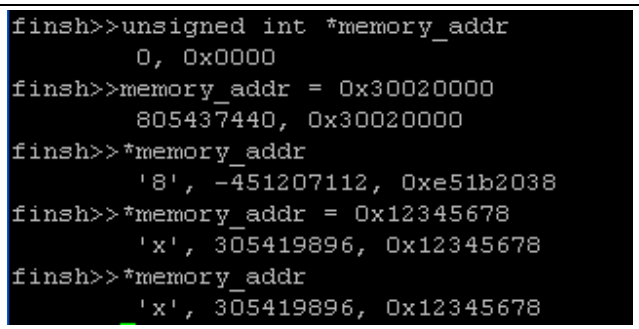

图 9-18 在 Finsh Shell 中解释运行代码

如图 9-18 所示，每条代码执行完，下面会打印出一系列的数字，这些代表的是执行这些表达式的结果，例如 memory_addr = 0x12345678; 执行完会把 memory_addr 的值显示出来。执行*memory_addr，得到返回值 0xe51b2038，表示 memory_addr 指向的内存地址中值是 0xe51b2038。执行*memory_addr = 0x12345678 后，memory_addr 指向的的内存地址 0x30200000 被写成 0x12345678。扩展出去，通过这种方式，也可以将开发板上 LED 灯相应的点亮和熄灭。

(3) 能够直接在 shell 中调用系统函数，访问系统变量

例如，你在 LCD 驱动中实现了一个画线的函数 void hline(rt_uint32_t c, int x1, int x2, int y)；你只需在函数体外添加一条语句 FINSH_FUNCTION_EXPORT(hline, draw a hline);就可以将 hline 函数导出到 Finsh 命令中。然后你可以通过 Finsh shell 输入 hline(255, 10, 100, 50)，回车，屏幕上就会画出一条直线。系统变量的用法也是类似。此外，Finsh shell 还可以用来作为系统启动脚本；与上位机交互的接口；更可以用来做函数级白盒测试。总之，有了 Finsh shell，Nothing is impossible!

10 RT-Thread 内核实现及应用

有关 RT-Thread 内核实现以及应用,移植部分内容，请阅读 RT-Thread 工作室编写的文档<<RT-Thread 编程指南>>，该编程指南是 RT-Thread 开发人员必备宝典，其中介绍了 RT-Thread 的内部实现机制，编程接口，应用例程以及移植到其他平台的详细步骤，相信它可以真正带您进入 RT-Thread 的大门，图 10-1 所示为该编程指南的目录。

图 10-1 RT-Thread 编程指南目录

11 RT-Thread 相关常见问题

11.1 RT-Thread 从哪里而来？

RT-Thread RTOS，Kernel 部分完成于 2006 年上半年，创始人源于国内一老牌 RTOS：DOOLOO RTOS，甚至是 BSP 一些结构都源于 DOOLOO RTOS。但与 DOOLOO RTOS 明显不同的是，Kernel 完全重新编写，突出的是实时性和小而灵活，并且引入了内核的对象模型以摒弃内核对象的与动态内存管理器无关化。

11.2 RT-Thread 用于商业产品&工程，版权如何界定？

RT-Thread RTOS 内核部分完全由 RT-Thread 工作室编写，无其他版权问题，可以放心在商业产品 & 工程中使用。对于把 RT-Thread 使用于商业产品中，RT-Thread 工作室承诺永久不收费（使用人拥有使用权，使用用途责任请自行承担）。另外有两点需要注意：

(1) RT-Thread RTOS 代码原始版权属于 RT-Thread 所有。

(2) 在商业产品和工程中使用 RT-Thread RTOS，请在产品说明书上明确说明使用了 RT-Thread，如有串口输出，请在系统启动显示 RT-Thread 的版本信息。如使用了 RT-Thread RTGUI，请保留 RT-Thread LOGO。

11.3 RT-Thread RTOS 由谁开发，由谁维护？

目前 RT-Thread RTOS 由国内 RT-Thread 工作室开发及维护，官方网站是：http://www.rt-thread.org，上面会有 RT-Thread 的最新开发进展信息。

11.4 RT-Thread RTOS 是否已在产品中使用？稳定度和 BUG 情况如何？

目前已经有数家公司使用 RT-Thread RTOS 做为他们的系统平台，在上面进行产品开发，稳定性表现不错。就如同没有 100%的完美事物一样，BUG 是存在的，反馈上来 RT-Thread 工作室会努力尽快修正。

11.5 我能加入到 RT-Thread 的开发者队伍中吗？

能！
RT-Thread 工作室欢迎任何对 RTOS 感兴趣的人，不管你是学生或资深嵌入式系统开发工程师。RT-Thread 的开发人员通常依赖于论坛、邮件、GTalk 进行联系交流，由于目前上海的开发人员比较多一些，所以会不定期的在上海举行开发者聚会。

11.6 RT-Thread 移植性如何，容易移植到其他平台吗？

容易!
请阅读《RT-Thread 编程指南》，其中有移植到其他平台的详细步骤介绍。

11.7 我是新手，想学习 RT-Thread，该怎样入门呢？

请阅读《RT-Thread 编程指南》，同时阅读 RT-Thread 的核心代码，RT-Thread 的实现条理性是非常清晰的，且代码行数不多，不过需要些预备知识，主要是 list 链表的实现和对象的实现，对应的代码在 kservice.h 和 object.c 中，基础知识在编程指南中有提及。

11.8 RT-Thread 依靠什么持续发展下去，能够盈利吗？

目前 RT-Thread 的发展主要依赖于大家的兴趣爱好，大多数都是在业余时间进行开发的。以后会通过技术支持、组件定制、组件开发、辅助工具等方式进行盈利。从几大开源软件来看，商业支持是软件持续发展不可或缺的一部分，所以 RT-Thread 工作室希望能够有更多的公司选择 RT-Thread RTOS 做为系统平台，这个对于公司、对于整个 RT-Thread 社区都是双赢的局面。对于公司，能够获得免费的 RTOS 套件，同时也能够推动着这个 RTOS 套件不断的朝着稳定的方向发展。对于 RT-Thread 工作室，有公司支持的发展无疑会令 RT-Thread 的发展更上一层楼，当然也意味着以后的支持费用有着落啦。

12 RT-Thread 相关建议或技术支持

RT-Thread 官方网站：
http://www.rt-thread.org
RT-Thread 官方网站论坛：
http://www.rt-thread.org/phpbb
RT-Thread 英文网站及 SVN 源代码服务器：
http://rt-thread.googlecode.com
阿莫综合电子网站 RT-Thread 专版：
http://www.ourdev.cn/bbs/bbs_list.jsp?bbs_id=3066