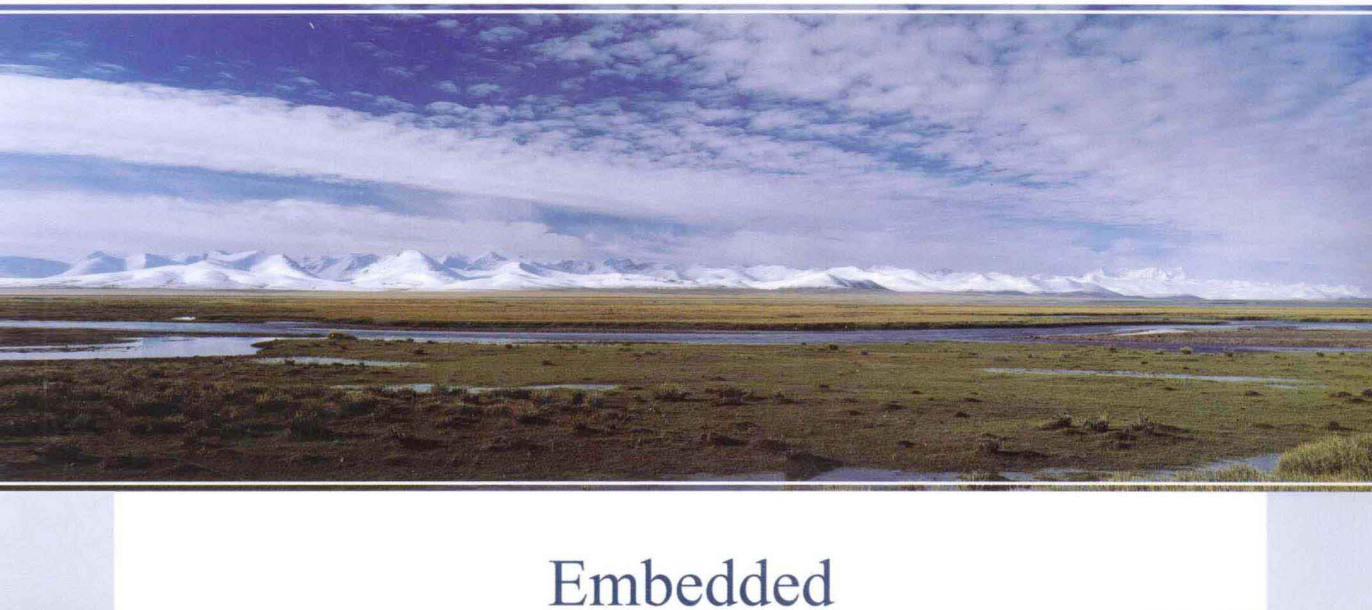
高等院校嵌入式人才培养规划教材
Gaodeng Yuanxiao Qianrushi Rencai Peiyang Guihua Jiaocai

嵌入式应用程序设计综合教程

华清远见嵌入式学院 曾宏安 冯利美 主编



Embedded Application Design

典型嵌入式开发项目
嵌入式程序设计综合实训
配备实训项目拓展和经典操作视频

DVD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

高等院校嵌入式人才培养规划

Gaodeng Yuanxiao Qianrushi Rencai Peiyang Guihua

嵌入式应用程序设计综合教程

华清远见嵌入式学院 曾宏安 冯利美 主编

Embedded
Application Design

人 民 邮 电 出 版 社

图书在版编目（CIP）数据

嵌入式应用程序设计综合教程 / 曾宏安，冯利美主编. -- 北京 : 人民邮电出版社，2014.2
高等院校嵌入式人才培养规划教材
ISBN 978-7-115-33000-0

Ⅰ. ①嵌… Ⅱ. ①曾… ②冯… Ⅲ. ①Linux操作系统－高等学校－教材 Ⅳ. ①TP316.89

中国版本图书馆CIP数据核字(2013)第204632号

内 容 提 要

本书结合大量实例，讲解了嵌入式应用程序设计各个方面的基本方法，以及必要的核心概念。主要内容包括搭建嵌入式 Linux 开发环境、标准 I/O 编程、文件 I/O 编程、进程控制开发、进程间通信、多线程编程、嵌入式 Linux 网络编程等。重视应用是贯穿全书的最大特点，本书在各章和全书结尾分别设置了在项目实践中常见和类似的应用实例。

本书可以作为高等院校电子、通信、计算机、自动化等专业的嵌入式 Linux 开发课程的教材，也可供嵌入式开发人员参考。学习本书应具有 Linux C 语言编程的基本知识。

◆ 主　　编　华清远见嵌入式学院　曾宏安　冯利美
责任编辑　王　威
责任印制　杨林杰

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京中新伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：13.5　　2014 年 2 月第 1 版
字数：328 千字　　2014 年 2 月北京第 1 次印刷

定价：39.80 元（附光盘）

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

前　言

随着消费群体对产品要求的日益提高，嵌入式技术在消费类电子、智能家电、通信、汽车电子、医疗设备等领域得到了广泛的应用。企业对嵌入式人才的需求也越来越多。近几年来，各高等院校和高等职业院校开始纷纷开设嵌入式专业或方向。与此同时，各院校在嵌入式专业教学建设的过程中几乎都面临教材难觅的困境。虽然目前市场上的嵌入式开发相关书籍比较多，但几乎都是针对有一定基础的行业内研发人员而编写的，并不完全符合学校的教学要求。学校教学需要一套充分考虑学生现有知识基础和接受能力，明确各门课程教学目标的，便于学校安排课时的嵌入式专业教材。

针对教材缺乏的问题，我们以多年来在嵌入式工程技术领域内人才培养、项目研发的经验为基础，汇总了近几年积累的数百家企业对嵌入式研发相关岗位的真实需求，调研了数十所开设“嵌入式工程技术”专业的高等院校和高等职业院校的课程设置情况、学生特点和教学用书现状。通过细致的整理和分析，对专业技能和基本知识进行合理划分，我们在 2009 年编写了一套高等院校嵌入式人才培养规划教材，包括以下 5 种。

《嵌入式技术基础》

《ARM 嵌入式体系结构与接口技术》

《嵌入式 Linux 操作系统》

《嵌入式 Linux C 语言开发》

《嵌入式应用程序设计》

经过 4 年，嵌入式行业发生了巨大的变化，产品升级换了代，而高校中的嵌入式专业也日驱成熟，有些教材已无法满足新的需要，有的新开的课程缺少配套教材，所以本次又增补了几个新品种。

本书是其中之一，全书共 7 章，内容涵盖嵌入式 Linux 应用开发的主要方面。

第 1 章介绍 Linux 标准 I/O 编程，让读者了解用户编程接口（API）和系统调用之间的关系并掌握基本的文件访问方法。

第 2 章介绍 Linux 文件 I/O 编程，分析了标准 I/O 和文件 I/O 的区别，重点讲解文件描述符的含义和具体的文件 I/O 编程接口。

第 3 章介绍 Linux 多任务机制，主要讲解了 Linux 中进程和线程的区别和联系、如何创建多进程以及守护进程。

第 4 章介绍 Linux 进程间通信，主要讲解了几种常用的进程通信方法，包括管道通信、信号通信、共享内存、消息队列等。

第 5 章介绍 Linux 多线程编程，主要讲解了 Linux 环境下的多线程编程方法及注意事项。

第 6 章介绍 Linux 网络编程，主要讲解了 Linux 环境下的网络编程方法，涉及网络体系结构、TCP 编程、UDP 编程和服务器模型等。

第 7 章介绍 Linux 高级网络编程，主要讲解网络超时检测、广播、组播和 UNIX 域套接字的基本编程方法。

本书由曾宏安、冯利美主编并统校全稿。本书的完成需要感谢华清远见嵌入式学院，教材内容参考了学院与嵌入式企业需求无缝对接的、科学的专业人才培养体系。同时，嵌入式学院从业或执教多年的行业专家团队也对教材的编写工作做出了贡献，刘洪涛、冯利美、曹忠明、赵孝强、程姚根、季久峰、贾燕枫、关晓强等老师在书稿的编写过程中认真阅读了所有章节，提供了大量在实际教学中积累的重要素材，对教材结构、内容提出了中肯的建议，并在后期审校工作中提供了很多帮助，在此表示衷心的感谢。

本书所有源代码、PPT 课件、教学素材等辅助教学资料，请到人民邮电出版社教学服务与资源网（www.ptpedu.com.cn）下载。

由于作者水平所限，书中不妥之处在所难免，恳请读者批评指正。对于本书的批评和建议，可以发到 www.embedu.org 技术论坛。

编　者

2013 年 6 月

目 录

第1章 Linux 标准 I/O 编程

在应用开发中经常要访问文件。Linux 下读写文件的方式有两大类：标准 I/O 和文件 I/O。其中标准 I/O 是最常用也是最基本的内容，希望读者好好掌握。

本章主要内容：

- Linux 系统调用和用户编程接口（API）；
- Linux 标准 I/O 概述；
- 标准 I/O 操作。

1.1 Linux 系统调用和用户编程接口

1.1.1 系统调用

操作系统负责管理和分配所有的计算机资源。为了更好地服务于应用程序，操作系统提供了一组特殊接口——系统调用。通过这组接口用户程序可以使用操作系统内核提供的各种功能。例如分配内存、创建进程、实现进程之间的通信等。

为什么不允许程序直接访问计算机资源？答案是不安全。单片机开发中，由于不需要操作系统，所以开发人员可以编写代码直接访问硬件。而在 32 位嵌入式系统中通常都要运行操作系统，程序访问资源的方式就发生了改变。操作系统基本上都支持多任务，即同时可以运行多个程序。如果允许程序直接访问系统资源，肯定会带来很多问题。因此，所有软硬件资源的管理和分配都由操作系统负责。程序要获取资源（如分配内存，读写串口）必须通过操作系统来完成，即用户程序向操作系统发出服务请求，操作系统收到请求后执行相关的代码来处理。

用户程序向操作系统提出请求的接口就是系统调用。所有的操作系统都会提供系统调用接口，只不过不同的操作系统提供的系统调用接口各不相同。Linux 系统调用接口非常精简，它继承了 UNIX 系统调用中最基本和最有用的部分。这些系统调用按照功能大致可分为进程控制、进程间通信、文件系统控制、存储管理、网络管理、套接字控制、用户管理等几类。

1.1.2 用户编程接口

前面提到利用系统调用接口程序可以访问各种资源，但在实际开发中程序并不直接使用系统调用接口，而是使用用户编程接口（API）。为什么不直接使用系统调用接口呢？原因如下。

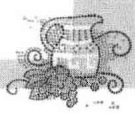
（1）系统调用接口功能非常简单，无法满足程序的需求。

（2）不同操作系统的系统调用接口不兼容，程序移植时工作量大。

用户编程接口通俗的解释就是各种库（最重要的就是 C 库）中的函数。为了提高开发效率，C 库中实现了很多函数。这些函数实现了常用的功能，供程序员调用。这样一来，程序员不需要自己编写这些代码，直接调用库函数就可以实现基本功能，提高了代码的复用率。使用用户编程接口还有一个好处：程序具有良好的可移植行。几乎所有的操作系统上都实现了 C 库，所以程序通常只需要重新编译一下就可以在其他操作系统下运行。

用户编程接口（API）在实现时，通常都要依赖系统调用接口。例如，创建进程的 API 函数 fork()对应于内核空间的 sys_fork()系统调用。很多 API 函数需要通过多个系统调用来完成其功能。还有一些 API 函数不需要调用任何系统调用。

在 Linux 中用户编程接口（API）遵循了在 UNIX 中最流行的应用编程界面标准——POSIX 标准。POSIX 标准是由 IEEE 和 ISO/IEC 共同开发的标准系统。该标准基于当时现有的 UNIX



实践和经验，描述了操作系统的系统调用编程接口（实际上就是 API），用于保证应用程序可以在源代码一级上在多种操作系统上移植运行。这些系统调用编程接口主要是通过 C 库（libc）实现的。

1.2 Linux 标准 I/O 概述

1.2.1 标准 I/O 的由来

标准 I/O 指的是 ANSI C 中定义的用于 I/O 操作的一系列函数。

只要操作系统中安装了 C 库，标准 I/O 函数就可以调用。换句话说，如果程序中使用的是标准 I/O 函数，那么源代码不需要修改就可以在其他操作系统下编译运行，具有更好的可移植性。

除此之外，使用标准 I/O 可以减少系统调用的次数，提高系统效率。标准 I/O 函数在执行时也会用到系统调用。在执行系统调用时，Linux 必须从用户态切换到内核态，处理相应的请求，然后再返回到用户态。如果频繁地执行系统调用会增加系统的开销。为了避免这种情况，标准 I/O 使用时在用户空间创建缓冲区，读写时先操作缓冲区，在合适的时机再通过系统调用访问实际的文件，从而减少了使用系统调用的次数。

1.2.2 流的含义

标准 I/O 的核心对象就是流。当用标准 I/O 打开一个文件时，就会创建一个 FILE 结构体描述该文件（或者理解为创建一个 FILE 结构体和实际打开的文件关联起来）。我们把这个 FILE 结构体形象地称为流。标准 I/O 函数都基于流进行各种操作。

标准 I/O 中的流的缓冲类型有以下三种。

（1）全缓冲：在这种情况下，当填满标准 I/O 缓冲区后才进行实际 I/O 操作。对于存放在磁盘上的普通文件用标准 I/O 打开时默认是全缓冲的。当缓冲区已满或执行 flush 操作时才会进行磁盘操作。

（2）行缓冲：在这种情况下，当在输入和输出中遇到换行符时执行 I/O 操作。标准输入流和标准输出流就是使用行缓冲的典型例子。

（3）无缓冲：不对 I/O 操作进行缓冲，即在对流的读写时会立刻操作实际的文件。标准出错流是不带缓冲的，这就使得出错信息可以立刻显示在终端上，而不管输出的内容是否包含换行符。

在下面讨论具体函数时，请读者注意区分以上的 3 种不同情况。

1.3 标准 I/O 编程

本节所要讨论的 I/O 操作都是基于流的，它符合 ANSI C 的标准。有一些函数读者已经非常熟悉了（如 printf()、scantf()函数等），因此本节中仅介绍最常用的函数。

1.3.1 流的打开

使用标准 I/O 打开文件的函数有 fopen()、fdopen()和 freopen()。它们可以以不同的模式打开文件，都返回一个指向 FILE 的指针，该指针指向对应的 I/O 流。此后，对文件的读写都是通过这个 FILE 指针来进行。其中 fopen()可以指定打开文件的路径和模式，fdopen()可以指定打开的文件描述符和模式，而 freopen()除可指定打开的文件、模式外，还可指定特定的 I/O 流。

fopen()函数格式如表 1.1 所示。

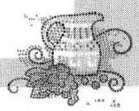
表 1.1　　fopen()函数语法要点

所需头文件	#include <stdio.h>
函数原型	FILE * fopen（const char * path, const char * mode）;
函数参数	path：包含要打开的文件路径及文件名
	mode：文件打开方式，详细信息参考表 1.2
函数返回值	成功：指向 FILE 的指针
	失败：NULL

其中，mode 用于指定打开文件的方式。表 1.2 说明了 fopen()中 mode 的各种取值。

表 1.2　　mode 取值说明

r 或 rb	打开只读文件，该文件必须存在
r + 或 r + b	打开可读写的文件，该文件必须存在
w 或 wb	打开只写文件，若文件存在则文件长度为 0，即会擦写文件以前的内容；若文件不存在则建立该文件
w+或 w + b	打开可读写文件，若文件存在则文件长度为 0，即会擦写文件以前的内容；若文件不存在则建立该文件
a 或 ab	以附加的方式打开只写文件。若文件不存在，则会建立该文件；如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留
a+或 a + b	以附加方式打开可读写的文件。若文件不存在，则会建立该文件；如果文件存在，写入的数据会被加到文件尾后，即文件原先的内容会被保留



注意：在每个选项中加入 b 字符用来告诉函数库打开的文件为二进制文件，而非纯文本文件。不过在 Linux 系统中会忽略该符号。

当用户程序运行时，系统自动打开了三个流：标准输入流 stdin、标准输出流 stdout 和标准错误流 stderr。stdin 用来从标准输入设备（默认是键盘）中读取输入内容；stdout 用来向标准输出设备（默认是当前终端）输出内容；stderr 用来向标准错误设备（默认是当前终端）输出错误信息。

1.3.2　流的关闭

关闭流的函数为 fclose()，该函数将流的缓冲区内的数据全部写入文件中，并释放相关资源。fclose()函数格式如表 1.3 所示。

表 1.3　　fclose()函数语法要点

所需头文件	#include <stdio.h>
函数原型	int fclose（FILE * stream）;
函数参数	stream：已打开的流指针
函数返回值	成功：0
	失败：EOF

程序结束时会自动关闭所有打开的流。

1.3.3　错误处理

标准 I/O 函数执行时如果出现错误，会把错误码保存在 errno 中。程序员可以通过相应的函数打印错误信息。

错误处理相关函数 perror 如表 1.4 所示。

表 1.4　　perror()函数语法要点

所需头文件	#include <stdio.h>
函数原型	void　perror（const char* s）;
函数参数	s：在标准错误流上输出的信息
函数返回值	无

```
#include <stdio.h>
int main()
{
    FILE *fp;  //  定义流指针
    if ((fp = fopen("1.txt", "r")) == NULL)  //  NULL是系统定义的宏，其值为0
    {
        perror("fail to fopen");  //  输出错误信息
        return -1;
    }
    fclose(fp);
    return 0;
}
```

如果文件 1.txt 不存在，程序执行时会打印如下信息：

fail to fopen: No such file or directory

错误处理相关函数 strerror 如表 1.5 所示。

表 1.5　　strerror()函数语法要点

所需头文件	#include <string.h> #include <errno.h>
函数原型	char *strerror（int errnum）;
函数参数	错误码
函数返回值	错误码对应的错误信息

```
#include <stdio.h>
int main()
{
    FILE *fp;
    if ((fp = fopen("1.txt", "r")) == NULL)
    {
        printf("fail to fopen: %s\n", strerror(errno));
        return -1;
    }
    fclose(fp);
    return 0;
}
```

如果文件 1.txt 不存在，程序执行时会打印如下信息：

fail to fopen: No such file or directory

1.3.4　流的读写

1. 按字符（字节）输入/输出

字符输入/输出函数一次仅读写一个字符。其中字符输入/输出函数如表 1.6 和表 1.7 所示。

表 1.6　　字符输入函数语法要点

所需头文件	#include <stdio.h>
函数原型	int getc（FILE * stream）； int fgetc（FILE * stream）； int getchar（void）；
函数参数	stream：要输入的文件流
函数返回值	成功：读取的字符
	失败：EOF

getc()和 fgetc ()从指定的流中读取一个字符（节），getchar()从 stdin 中读取一个字符（节）。

表 1.7　　字符输出函数语法要点

所需头文件	#include <stdio.h>
函数原型	int putc（int c, FILE * stream）； int fputc（int c, FILE * stream）； int putchar（int c）；
函数返回值	成功：输出的字符 c
	失败：EOF

putc()和 fputc()向指定的流输出一个字符（节），putchar()向 stdout 输出一个字符（节）。

下面这个实例结合 fputc()和 fgetc()，循环从标准输入读取任意个字符并将其中的数字输出到标准输出。

```
/*fput.c*/
#include <stdio.h>
int main()
```

```
{
    int c;
    while ( 1 )
    {
       c = fgetc(stdin); // 从键盘读取一个字符
       if ((c >= '0') && (c <= '9')) fputc(c, stdout); // 若输入的是数字，输出
       if (c == '\n') break; // 若遇到换行符，跳出循环
    }
    return 0;
}
```

运行结果如下。

```
$ ./a.out
abc98io#4/wm
984
```

2. 按行输入/输出

行输入/输出函数一次操作一行。其中行输入/输出函数如表 1.8 和表 1.9 所示。

表 1.8　　行输入函数语法要点

所需头文件	#include <stdio.h>
函数原型	char * gets（char *s）
	char * fgets（char * s, int size, FILE * stream）
函数参数	s：存放输入字符串的缓冲区首地址
	size：输入的字符串长度
	stream：对应的流
函数返回值	成功：s
	失败或到达文件末尾：NULL

gets 函数容易造成缓冲区溢出，不推荐大家使用。

fgets 从指定的流中读取一个字符串，当遇到\n 或读取了 size-1 个字符后返回。注意，fgets 不能保证每次都能读出一行。

表 1.9　　行输出函数语法要点

所需头文件	#include <stdio.h>
函数原型	int puts（const char *s）
	int fputs（const char * s, FILE * stream）
函数参数	s：存放输出字符串的缓冲区首地址
	stream：对应的流
函数返回值	成功：s
	失败：NULL

下面以 fgets()为例计算一个文本文件的行数。

```
/*fgets.c*/
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
    int line = 0;
    char buf[128];
    FILE *fp;

    if (argc < 2)
    {
        printf("Usage : %s <file>\n", argv[0]);
        return -1;
    }
    if ((fp = fopen(argv[1], "r")) == NULL)
    {
        perror("fail to fopen");
        return -1;
    }
    while(fgets(buf, 128, fp) != NULL)
    {
        if (buf[strlen(buf)-1] == '\n') line++;
    }
```