基于嵌入式 Linux 的 RFID 安检系统的设计

王志亮,官洪运,王 龙,刘 婕,张智轶
(东华大学 信息学院,上海 201620)

摘 要:介绍了系统的整体框架和 Linux 下串口通信开发的一般流程,着重介绍了嵌入式环境下数据库 sqlite3 的移植和操作,并给出了串口与 sqlite3 数据库通信的效率和稳定性测试,在此基础上分析了基于嵌入式 Linux 串口通信的 RFID 安检的可行性,在基于 ARM9 硬件平台、Linux 操作系统环境下实现了 RS232 串口与 RFID 读写器通信的应用开发。

关键词:嵌入式;Linux;RFID;串口通信;sqlite3
中图分类号:TN92 **文献标识码**:A **文章编号**:1674-7720(2010)22-0092-04

Design of RFID safety inspection system based on embedded Linux OS

WANG Zhi Liang, GUAN Hong Yun, WANG Long, LIU Jie, ZHANG Zhi Yi
(College of Information, Donghua University, Shanghai 201620, China)

Abstract: This paper introduces the overall framework of the system and the general development process of serial communication in Linux OS, and then focuses on the transplantation and operations of sqlite3 database. At last it gives the efficiency and stability testing, and analyses the feasibility of RFID safety inspection system based on serial communication in embedded Linux OS. Based on ARM9 hardware platform and Linux operating system Environment, this article achieves an application development of RS232 serial communication with the RFID reader.

Key words: embedded; Linux; RFID; serial communication; sqlite3

RFID(射频识别)是一种非接触式的自动识别技术,它通过射频信号自动识别目标对象并获取相关数据,识别工作无需人工干预,可工作于各种恶劣环境下。RFID 技术可识别高速运动物体并可同时识别多个标签,操作快捷方便。非接触 IC 卡是目前 RFID 系统中最常用的一种电子标签,它诞生于 20 世纪 90 年代初,是世界上最近几年发展起来的一项新技术,它成功地将射频识技术和 IC 卡技术结合起来,解决了无源(卡中无电源)和免接触这一难题,是电子器件领域的一大突破。由于存在着磁卡和接触式 IC 卡不可比拟的优点,使之一经问世,便立即引起广泛的关注,并以惊人的速度得到推广应用,如我国的第二代公民身份证、公交卡、ETC 免停车付费卡等。可以说 RFID 技术越来越多地应用到我国身份安检、质量安检、车辆安检、执法安检等诸多安检系统中。

本文就是针对安检系统这种工程背景下 RFID 通信的应用开发。现在一般的 RFID 通信都基于串口,串口因其通用性、方面性和优良性能得到了广泛的应用。由于安检系统中往往涉及大量重要数据的读取、通信以及实时更新,因此数据库技术的引入必不可少。同时本文选择了 Linux 操作系统,众所周知 Linux 同 Windows 相比性能更安全、更可靠,而且 Linux 还是一款免费的代码开源的操作系统,裁减内核更方便、快捷,与其他操作系统相比有着许多独特的优势,更加适合用作嵌入式操作系统。

1 系统结构介绍

RFID 安检系统主要包括 RFID 前段读写器、嵌入式 Linux 终端两大部分,其结构如图 1 所示。

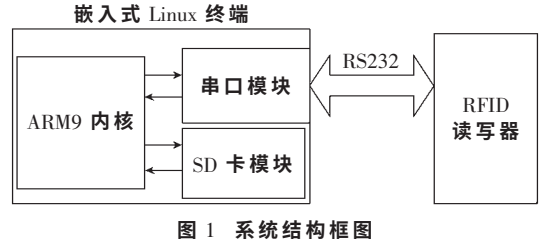


图 1 系统结构框图

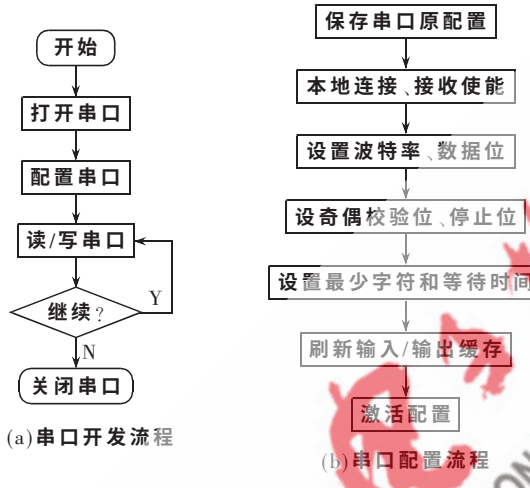
其中嵌入式终端的 CPU 采用 ARM9 内核,内核执行速率达几百兆赫兹,可以很好地满足 RFID 数据的读取和存储。由于嵌入式系统一般是一个经过裁剪、资源极

其有限的系统，因此对于安检系统中涉及到的大量数据只能存取到外围存储设备中，本方案中的SD卡模块正是用来存储数据库的，当RFID读写器读取到指定数据，便在SD卡中的相关数据库文件中查询，并根据查询结果做出相关反应并及时更新本地数据库。

2 Linux下串口的开发

在Linux下对串口进行配置、打开、读写等一系列的操作其使用方式与文件操作一样，区别在于串口是一个终端设备[1]。Linux中的串口设备文件存放于/dev目录下，其中串口1、串口2一般对应设备名依次为“/dev/ttyS0”、“/dev/ttyS1”。在使用串口之前必须设置相关配置，包括波特率、数据位、校验位、停止位等[2]。

串口开发的流程和串口开发过程中串口配置的流程分别如图2(a)、图2(b)所示。



(a)串口开发流程　(b)串口配置流程

图2 串口开发与配置流程

串口设置由下面结构体实现：

```
Struct termios {
tcflag_t  c_iflag;        /* input flags */
tcflag_t  c_oflag;        /* output flags*/
tcflag_t  c_cflag;        /*control flags */
tcflag_t  c_lflag;        /* local flags */
tcflag_t  c_cc[NCSS];     /* control characters */
}
```

按照串口配置流程，对termios结构体设置相关参数，当串口按自己的设置要求配置成功后，即可将串口当做普通I/O文件，使用read和write函数对串口进行读取。

3 sqlite3数据库的应用开发

sqlite3数据库是一种嵌入式数据库，其目标是尽量简单，因此抛弃了传统企业级数据库的种种复杂特性，只实现对于数据库而言必备的功能。尽管简单性是sqlite3追求的首要目标，但是其功能和性能都非常出色，具有支持SQL92标准、所有数据存放到单独的文件中支持的最大文件可达2 TB、数据库可以在不同字节的机器之间共享、体积小、系统开销小、检索效率高、支持多种计算机语言、源码开放，并且可以用于任何合法用途等特性。

3.1 sqlite3数据库的移植

sqlite3数据库的移植过程如下所述：

(1)首先从sqlite官网上下载最新的sqlite3源码包；

(2)解压源码包，并进入解压目录：

```
tar -zxvf sqlite-3.6.23.1.tar.gz
cd sqlite-3.6.23.1
```

(3)配置Configure脚本，使用相关选项生成编译文件Makefile文件：

```
./configure - -enable-share - -prefix=./sqlite-3.6.23.1/result - -host=arm-linux
```

选项-enable-share指定使用Linux的共享库

选项-prefix指定了安装目录为./sqlite-3.6.23.1/result

选项-host指定了编译环境为目标机为arm的交叉编译环境

(4)交叉编译，生成嵌入式终端下数据库的管理程序和库文件，最终在result目录下得到数据库管理程序sqlite3(相当于Windows下Access程序)，提供编程所需的API的动态库libsqlite3.so.0.8.6，编程所需的头文件sqlite3ext.h sqlite3.h。交叉编译的命令如下：

```
Make
Make install
```

(5)将数据库管理程序sqlite3、提供编程所需的API的动态库libsqlite3.so.0.8.6及其1个软链接拷贝到开发板根文件系统相应位置，分别在嵌入式终端的/usr/bin和/usr/lib这两个目录下，命令如下：

```
Cp result/bin/sqlite3  /arm-linux/usr/bin
Cp  -l result/lib/libsqlite3.so*  /arm-linux/usr/lib
```

(6)为了能在开发机上编译，调用了sqlite3数据库的API的应用程序，需要将动态库libsqlite3.so.0.8.6及其2个软链接、2个头文件拷贝到交叉编译工具链所在目录的适当位置，至此sqlite3数据库的移植和开发环境的配置已完成。只要输入SQL语言便可以进行相关操作。

3.2 Linux下sqlite3的C语言开发

sqlite3里最常用到的是sqlite3 *类型。从数据库打开时开始，sqlite3就要为这个类型准备好内存，直到数据库关闭，整个过程都需要用到这个类型。数据库打开时起，这个类型的变量就代表了所要操作的数据库。

(1)打开数据库API接口函数

```
int sqlite3_open(文件名, sqlite3 *);
```

用这个函数开始数据库操作。需要传入两个参数，其中之一是数据库文件名，例如：/home/test.db文件名不需要一定存在，如果此文件不存在，sqlite3会自动建立；如果存在，就尝试把它当数据库文件打开。

sqlite3 *参数即前面提到的关键数据结构。函数返

回值表示操作是否正确，如果是 SQLITE_OK 则表示操作正常。相关的返回值 sqlite3 定义了一些宏，具体这些宏的含义可以参考 sqlite3.h 文件。

(2)关闭数据库 API 接口函数

int sqlite3_close(sqlite3 *);

如果前面用 sqlite3_open 开启了一个数据库，结尾时不要忘了用这个函数关闭数据库。

(3)执行 SQL 语句 API 接口

由于嵌入式 sqlite3 数据库支持 SQL 语言，因而调用 C 中相关执行函数就如同在终端下操作数据库一样方面快捷，下面是具体的 API 函数：

这就是执行一条 sql 语句的函数。

Int sqlite3_exec(sqlite3 * db, const char *sql,sqlite3_callback,Void * ,char ** errmsg);

参数 1 是调用打开数据库函数 sqlite3_open()打开的数据库对象。

参数 2 是一条待执行的 SQL 语句，其语法格式同标准 SQL 语言规范一样，如创建 table 时插入的记录如下：

create table student (id varchar (10) primary key, age smallint);

此语句创建了名为 student 的表，表中定义了 id(学号)和年纪两个变量，其中 id 是主键。

Insert into student values(12345678,21);

此语句向 student 表中插入一组数据(12345678,21)，其中学号为 12345678,学生年龄为 21。

对于数据库的其他操作，如数据库更新、修改、查找等用法同上。

参数 3 sqlite3_callback 是自定义的回调函数，对执行结果的每一行都执行一次这个函数。

参数 4 void * 是调用者所提供的指针，你可以传递任何一个指针参数到这里，这个参数最终会传到回调函数里，如果不需要传递指针给回调函数，可以填 NULL。

参数 5 char ** errmsg 是错误信息。sqlite3 里面有很多固定的错误信息。执行 sqlite3_exec 之后，如果执行失败则可以查阅这个指针，即可知道执行过程中错误发生的位置。

3.3 串口同 sqlite3 通信测试与分析

为了验证 sqlite3 数据库在嵌入式 Linux[3-4]终端下的执行效率和稳定性，为此做了一个简单的测试实验：通过上位机程序向嵌入式 Linux 终端的串口定时发送字符串；嵌入式 Linux 终端接收到字符串便立即写入到下位机的数据库中。自后查看数据中的数据，看看有没有遗漏和误码。上位机的程序使用 VC6.0 开发，整个程序界面只设了一个按键，按下按键，上位机就向嵌入式 Linux 终端不停地发送字符串数据,按键响应程序设计如下：

```
void CSendDlg::OnButton_Click()
{
  state=1;
  while(1)
  {
    str.Format("第%3d 条记录",state); //格式化字符串格式
    m_Port.WriteToPort(str,str.GetLength()); //向串口发送字符串
        state++;
    Sleep(100);                              //延时 100 ms
  }
```

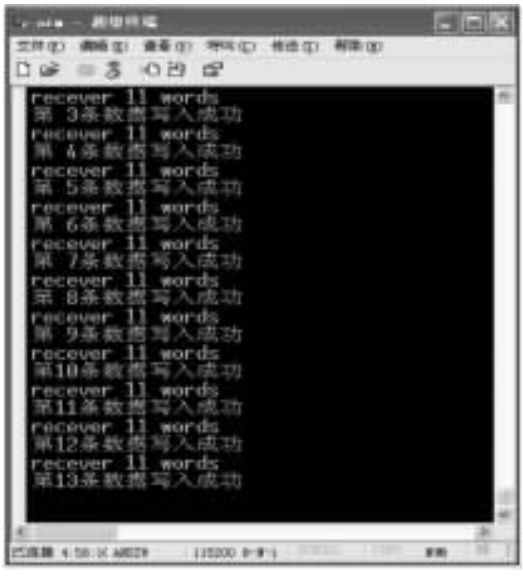
可见程序是个定时 100 ms 便发送一条字符串的循环，而且发送的每一条字符串事先通过 str.Format 格式化为固定长度，本例中是 11 B。按下按键后发送的第一条字符串为：“第 1 条记录”，每发送一条字符串里面的数字加“1”，这样写到数据库中就可以很清楚地查看有没有遗漏和误码，而且可以通过修改 Sleep 函数的延时参数检测出嵌入式 Linux 终端下 sqlite3 数据库操作的速度。

下位机嵌入式 Linux 终端的程序设计为：先创建一个数据库文件 test.db,接着就是一个死循环，串口不停地查找有没有数据写入，当检测到数据时，便写入到 test.db 中，若写入有误，则立即跳出循环，终止程序。

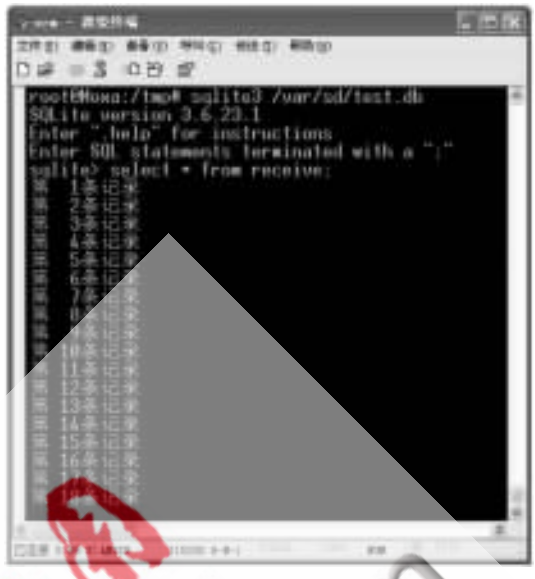
```
char sql[100]="create table receive(name varchar(40))";
qlite3_open("/var/sd/test.db",&db);          //在 SD 卡中创建
                                              test.db 文件
sqlite3_exec(db,sql,0,0,&errmsg);       //在 test.db 文件中插入
                                              表 receiver
fd=open_port(fd,1)                            //打开串口 1
set_opt(fd,9600,8,'N',1)                 //配置串口属性，开始通信
while(1)
{
    n=0;
    i=0;
    bzero(read_buf, sizeof(read_buf));
    if( (n=read(fd, read_buf, sizeof(read_buf))) <=0)
          Continue;          //未读到数据则继续查找串口
  printf("recever %d words\n",n);      //输出读到的字符数
    sprintf(sql,"insert into receive values(%s)",read_buf);
  result =sqlite3_exec(db,sql,0,0,&errmsg);       //插入数据
                                                  到数据库中
    if(result==SQLITE_OK)
     printf("第%3d 条数据写入成功 \n",++i);
                                          //若插入成功则提示
  else break;                    //若插入不成功，则跳出循环
}
```

测试结果如图 3 所示。

整个测试根据上位机串口发送的频率不同做了多组实验，每组实验写入 1 000 个数据，最终结果分析如下：上位机在定时 80 ms 左右或大于 80 ms 的情况下发送数据时，数据库写入的误码率为零；当定时时间小于 80 ms 时，随着定时时间变小误码率会越来越高。通过数



(a)下位机串口接收和数据库写数据提示



(b)查询 test.db 中数据

图 3 测试结果

据分析可知原因有以下几点:一是数据库本身写入需用时几十毫秒,二是 SD 卡并非高速读写设备,当数据还未完全写入数据库时若有新数据发过来,则下次读写将会发生难以估计的错误。实验还得出了当把数据库文件写入到系统 Flash 上的总耗时约为 50 ms,比写入 SD 卡中约少 30 ms。不过就 80 ms 左右的一次读写速度而言,嵌入式数据库 sqlite3 执行效率和稳定性非常可观,现在一般的 RFID 读写器通过串口执行一条指令的时间也需几十毫秒的时间,因而使用 sqlite3 数据库在执行速率和稳定性上对于安检系统中 RFID 读写数据的处理可以很好地达到要求,而且 sqlite3 还支持数据加密,安全性同样非常出色。

本文介绍了此 RFID 安检系统的硬件框架和软件设计,实现了 RFID 安检系统基于嵌入式 Linux 下的串口通信以及数据库的应用。最后通过实验证明并确定了其在速率、稳定性方面的可行性,对于当今大多数 RFID 安检系统的开发具有一定的参考价值。

参考文献

[1] WALL K,WASTON M.GUN/Linux 编程指南(第 2 版)[M].WHITIS M,王勇,译.北京:清华大学出版社,2002.

[2] 华清远见嵌入式中心.嵌入式 Linux C 语言应用程序设计[M].北京:人民邮电出版社,2008.

[3] FELDHOFER M.A proposal for an authentication protocol in a security layer for RFID smart tags.IEEE Proceedings of MELECON,2004.

[4] 华清远见嵌入式中心.嵌入式 Linux 系统开发标准教程[M].北京:人民邮电出版社,2009.

(收稿日期:2010-06-24)

作者简介:

王志亮,男,1986年生,硕士,主要研究方向:嵌入式开发,网络通信。