Linux 环境下基于 UDP 的 socket 编程浅析

吴佩贤

绍兴文理学院计算机系　浙江　312000

摘要： socket 适用于网络环境中的进程间通信，它已成为当前许多操作系统的网络 API，也是网络操作系统中必不可少的基础功能。尽管 UDP 无法像 TCP 一样提供可靠的数据传输，但 UDP 并不比 TCP 缺乏优越性。随着 Linux 操作系统和 Internet 的不断发展，Linux 网络环境下尤其是基于 UDP 的 socket 通信技术仍广为注目。文章介绍了 Linux 平台下的 socket 及其编程原理，并通过一个 Java 编写的基于 UDP 的客户/服务器程序，描述了网络中不同主机上的两个进程之间的 socket 通信机制。

关键词： socket；UDP；TCP；客户/服务器；进程；Java

0　引言

Linux 是一个诞生于 Internet 和 WWW 的产品，它具有稳定、简易、高效、兼容性好等特点，并且支持多种网络协议，如 Ipv4、Ipv6、X.25、IPX、NETBIOS、DDP 等。套接字（socket）是介于网络应用层和传输层之间的编程接口，套接字接口提供了访问下层通信协议的大量系统调用和相应的数据结构，进程在 Linux 上的网络通信过程就是使用套接字传输数据的过程。套接字在 UDP/IP 网络模型中的地位如图 1 所示。

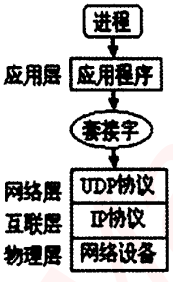

图1　套接字

1　BSD套接字接口

socket 最早应用于伯克莱大学 BSD UNIX 中，所以习惯上又称其为 BSD socket。一个套接字描述为一个通信连接的一端，在一个通信连接中的两端通信程序应各自有一个套接字来描述它们自己那一端，不同主机中的两个进程通过各自的套接字发送和接收消息，从而实现进程间跨网络的通信。

Linux 环境中套接字支持多种网络协议，不同类型网络协议具有不同的工作方式，所使用的地址格式也完全不同。对于各种网络协议而言，使用相同地址格式的几个协议称为一个协议地址簇，表 1 列出了 BSD 套接字的常见地址族。

表 1 BSD 套接字地址族的主要类型

地址族类型	对应的通信协议
AF_INET	TCP/IP 协议
AF_IPX	Novell IPX 协议
AF_UNIX	UNIX 内部套接字
AF_AX25	AX.25 协议套接字
AF_APPLETALK	APPLETALK DDS（Macintosh 机器用）

Linux 将套接字地址族抽象为统一的 BSD 套接字接口，该接口是应用程序的开发接口，由各地址族专有的软件支持。Linux BSD socket 支持以下常见套接字类型：

（1）SOCK_STREAM（数据流套接口）：提供一个面向连接的双工顺序数据流传输和可靠的数据传输服务。这种套接字可以保证数据传输的可靠性，不会出现数据丢失、破损或重复出现等差错，而且通过流量控制避免发送的数据流超限。Internet 地址中的 TCP 协议支持流套接字。

（2）SOCK_DGRAM（数据报套接口）：提供一个无连接和不可靠的双工数据传输服务。数据包以独立包形式被发送和接收。不对数据的传输提供无错保证，即数据可能被丢失、破坏，也可能被重复接收。Internet 地址中的 UDP 协议支持这种套接字。

（3）SOCK_RAW（原始套接口）：这种类型的套接字允许对低层协议如 IP 或 ICMP 直接访问，可以直接填充 IP、TCP、UDP 或者 ICMP 的包头，发送用户自己定义的 IP 包或者 ICMP 包，主要用于协议的开发。

2　用户数据报协议 UDP 及无连接服务

Internet 给分布式应用程序提供两类服务：一是面向连接的服务，二是无连接服务。面向连接的服务由 TCP 提供，客户和服务器在彼此间发送数据之前要先建立三次握手，以确保发送端的数据最终按顺序完整无误地传送给接收端。无连接服务则由 UDP 提供，发送端和接收端在传送数据之前不需要进行握手，因而不会带入任何延迟。

在无连接模式下，客户程序没有把套接字固定连接到一个指定的远程端点上，而是在每次发送数据报时指定远程目的地。因而 UDP 具有相当的灵活性：客户程序可以在它要求发送时才决定和哪个服务器交互。从图 1 可以看出，UDP

通过套接字直接从应用程序进程得到消息，附上源和目标端口号，并把得到的数据段传递给网络层。网络层将数据封装到IP数据包里，并使用尽力服务方式将数据包传递给接收端。如果该数据包到达接收端，UDP通过端口号和IP目标地址将数据包中的内容传正确地送给接收端的套接字。UDP在传递数据时并不保持发送端和接收端间的连接状态，即不对接收和发送缓冲区、拥塞控制、数据到达顺序等参数进行跟踪，因而UDP传送的数据包可能出错甚至丢失，其可靠性无法得到保障。尽管如此，UDP在DNS解析、IP电话、实时视频会议、可存储的音频视频流等应用程序中仍大受欢迎，其原因在于：（1）相对于TCP，面向无连接的数据传送不带时间延迟，速度自然要快。（2）UDP不对发送端和接收端间的诸多参数进行跟踪，使得它有能力支持更多的处于活动状态的客户，灵活性强。（3）在数据段结构对比中可以发现，UDP的数据包头部开销比TCP小得多。（4）由于没有拥塞控制机制，UDP发送数据的速度仅仅受限于应用程序产生数据的速度、源端的性能（CPU、时钟速度等）以及物理设备可以达到的带宽。

3 Linux平台下的socket编程原理

在Linux中，socket属于文件系统的一部分，网络通信可以被看作是对文件的读取。Linux的许多特性都非常有助于网络编程：首先，Linux拥有POSIX标准库函数，socket()、bind()、sendto()、recvfrom()等库函数可以方便地实现客户/服务器模型中数据的传送与接收。其次，Linux的进程管理策略也非常适合服务器的工作环境，Linux中的每个进程都对应一个父进程，同时它也能创建多个子进程。发送端的进程可创建多个子进程分别将各自的数据包发往接收端。第三，Linux传承了UNIX设备无关性这一优秀特性，即通过文件描述符实现了统一的设备接口：磁盘、显示终端、音频设备、打印设备甚至网络通信都使用统一的I/O调用。这些特性大大方便和简化了Linux环境下的网络程序设计。

3.1 socket函数

socket()用来创建套接口描述符，其格式声明为：

int socket（int domain，int type，int protocol）。参数domain说明网络程序所在的主机采用的通信协议，如AF_INET（IPv4协议）、AF_INET6（IPv6协议）、AF_LOCAL（Unix域协议）；参数type指定套接口类型（数据流套接口、数据报套接口、原始套接口），即相当于指明了网络程序所采用的通信协议（TCP还是UDP）；参数protocol由于指定了type，一般取0即可。

3.2 bind函数

格式声明为：

int bind（int sockfd，struct sockaddr *my_addr，int addrlen）。sockfd为套接口描述字；指针my_addr指向sockaddr结构，该结构包含了远程服务程序的IP地址与端口号；addrlen指明sockaddr结构的长度。

3.3 sendto函数

sendto()用于将应用缓冲区中的数据发送到指定的接收进程的地址空间，其格式声明为：

int sendto(int sockfd,const void *msg,int len,unsigned int flags,const struct sockaddr *to,int tolen)。参数sockfd用于指定UDP套接字描述符；msg是需要发送的数据缓冲区；to是指向包含目的IP地址和端口号的数据结构sockaddr的指针；tolen是地址的长度。若sendto发送数据成功，则返回发送数据量的长度，否则返回-1。

3.4 recvfrom函数

recvfrom()用于将套接字缓冲区中的数据接收并发给用户进程。其格式声明为：int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *addrlen)。参数sockfd指明要读取的套接字；buf指明应用层的接收缓冲区；len表示前面所示缓冲区的大小；from和addrlen则用于指定源地址和源端口号。

本文主要阐述的是无连接数据报通信的socket编程过程。客户/服务器主要编写客户程序和服务器程序，以下是主要的编程框架（图2）。

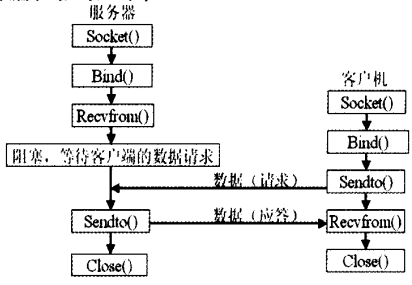

图2 UDP客户机/服务器程序的工作流程

无连接的数据报通信中，程序基本的编写步骤如图2所示。UDP通信的基本过程如下：在服务器端，先使用AF-INET协议族创建UDP数据报类型套接字，然后调用bind()给此套接字绑定一个端口。由于并不需要建立连接，因此服务器端可以通过调用recvfrom()在指定的端口等待客户端发送来的UDP数据报。而在客户端，先通过socket()创建一个数据报套接字，然后由操作系统为该套接字分配端口号。此后客户端就可以使用sendto()向一个指定的地址发送一个UDP套接字。在服务器端接收到套接字后，从recvfrom()中返回，在对数据报进行处理后，再用sendto()将处理的结

信时每次均需指定对端地址信息；也可以使用connect()来填写两端套接字的有关信息，然后直接使用recv()、send()进行通信。这样做的好处是，在随后通信程序中不必每次指定地址信息。套接口读写完毕后，通过close()调用关闭连接的套接口文件描述符。不论何种编程语言，和socket打交道都是这一组调用，只是在格式上有所区别。Visual Basic采用Winsock控件，C++ Builder采用TclientSocket和TserverSocket元件，Visual C++采用MFC类中的CAsyncSocket类和CSocket类，JAVA中通过Java.net.Socket和Java.net.ServerSocket类库来实现网络之间通信。

4 用Java编写的客户/服务器应用程序示例

网络应用程序的核心由一对程序组成，一个客户程序和一个服务器程序。当这两个程序执行的时候，系统就会创建一个客户端进程和一个服务器端进程，并且这两个进程通过对socket的读写来互相通信。下面就客户/服务器模型来对UDP的socket编程进行示范，程序功能如下：客户端从其键盘读入一行并将该行发送到通往服务器的套接字上；服务器从其套接字读入一行并将该行转换成大写；服务器将改写后的这一行输出到通往客户端的套接字上；客户端从它的套接字中读出修改后的行，并将该行在显示器上打印输出。

4.1 应用程序的客户端代码

```
import java.io.*;
import java.net.*;
class UDPClient
{
public static void main(String args[]) throws Exception
{
BufferedReader inFromUser = new BufferedReader (new InputStreamReader(System.in));
DatagramSocket clientSocket = new DatagramSocket();
InetAddress IPAddress = InetAddress.getByName ("hostname");
//hostname为服务器的主机名。通过getByName()将主机名解析为IP地址的DNS查找，并将得到的IP地址放入到IPAddress对象中
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
//定义两个字节数组，分别用来保留客户端发送和接收的数据
String sentence = inFromUser.readLine();
SendData = sentence.getBytes();
DatagramPacket sendPacket = new DatagramPacket
//构造分组sendPacket，客户端将借助它的套接字把该分组送入网络。该分组包括了分组sendData中所包含的数据、数据的长度、服务器的IP地址和应用程序的端口号（假定端口号为1234）
clientSocket.send(sendPacket);
//取得刚刚构造的分组，并通过clientSocket将它送入网络中。在发送分组之后，客户端就等待接收从服务器到来的分组
DatagramPacket receivePacket = new DatagramPacket (receiveData,receiveData.length);
//等待服务器发回分组的同时，客户端为该分组创建一个占位符receivePacket
clientSocket.receive(receivePacket);
//客户端保持空闲，直到接收到分组为止；当它接收到分组时，它就将分组放入receivePacket中
String modifiedSentence = new String(receivePacket.getData());
//将数据从receivePacket中抽取出来，并将字节数组转换成字符串modifiedSentence
System.out.println("Data From Server:" + modifiedSentence);
//在客户端的显示器上打印结果
clientSocket.close();
}
}
```

java.io和java.net是java的包，java.io包包含了输入和输出流的类，java.net包提供了网络支持类。该程序段构造了一个流inFromUser和一个DatagramSocket类型的套接字clientSocket。流inFromUser是程序的输入流，它被加到键盘上，当用户在键盘上打入字符的时候，字符就会进入到流inFromUser中。

4.2 应用程序的服务器端代码

```
import java.io.*;
import java.net.*;
class UDPServer
{
public start void main(String args[]) throws Exception
{
DatagramSocket serverSocket = new DatagramSocket (1234);
//在端口1234上构造了DatagramSocket类型的套接字serverSocket，所有发送和接收的数据都将通过该套接字进行传送
```

```
byte[] sendData = new byte[1024];
while(true)
{
 DatagramPacket receivePacket = new DatagramPacket
(receiveData,receiveData.length);
serverSocket.receive(receivePacket);
String sentence = new String(receivePacket.getData());
//从客户端传送过来的分组中把数据抽取出来，并将其
放入sentence中
InetAddress IPAddress = receivePacket.getAddress();
    //从客户端传送过来的分组中把IP地址抽取出来
int port = receivePacket.getPort();
//从客户端传送过来的分组中把客户端口号抽取出来
String newSentence = sentence.toUpperCase();
sendData = newSentence.getBytes();
DatagramPacket sendPacket = new DatagramPacket
(sendData,
sendData.length,IPAddress,port);
serverSocket.send(sendPacket);
}
}
}
```

将以上两个程序在各自的主机上分别安装并编译，基于UDP的基本原理，执行客户端的时候，客户进程不会发起与服务器的连接，因而可以首先执行客户端，然后再执行服务器端。一旦执行了客户和服务器程序，客户端的用户可以使用该应用程序来向服务器发送一行，并接收该行的转换成大写之后的字符串了。

5 结束语

Java是平台无关的，且编写的程序简洁。以上介绍了socket的编程机制及在Linux系统下的一个套接口程序，能够深入理解用socket机制和UDP工作模式，但怎样改进UDP中数据传送的可靠性仍是众多研究人员关注的方向。

参考文献

[1]毛德操,胡希明著.Linux内核源代码情景分析(下)[M].杭州：浙江大学出版社.2001.

[2]张威.Linux网络编程教程[M].北京：北京希望电子出版社.2002.

[3]Douglas.Comer，David L.Stevens.TCP/IP网络互联技术(卷3)[M].客户－服务器编程与应用.北京:清华大学出版社.2004.

[4]张义,李剑编.Java程序员开发开发指南[M].北京：北京希望电子出版社.2004.

Analysis of UDP-based Socket Programming on Linux

Wu Peixian

Computer Science Department,Shaoxing University,Zhejiang,312000

Abstract: Socket is suitable for communication between two processes from network. Now socket has been available network-API of many OS, at the same time it has been one necessary part of network-OS. Though UDP can,t provide reliable data transfer while TCP can, itís still as superior as TCP. UDP-based socket communication technology on Linux network environment is still wide concerned with Linux OS and Internetís developing. The paper introduces the socket and it,s programming principle on Linux. By using a UDP-based client/server program which is edited in Java, it describes socket communication mechanism between the two processes from the different computers in the network.

Keywords:socket;UDP;TCP;client/server;process;Java

上接68页

[2] Stephen Farrell,Russel Housley.An Internet Attribute Certificate Profile for Authority.IETF Internet Draft. May 2000.

[3] The Open Group.Authorization(AZN) API.January 2000.

[4] 王常杰,张方国,王育民.Internet移动代理技术中的安全性研究.西安电子科技大学学报.2001.

Attribute certificate based Privilege Management scheme for Mobile Agent System

Liu Haisheng,Qing Zikuo,Ma Wenping

Key Laboratory of Computer Network and Information security Ministry of Education,Xidian University,Sanxi,710071

Abstract: Mobile agent is a new technology for distributed computation which has a wide application.Privilege management is one of the significant techniques guaranteeing the security of mobile agents.This paper describes security in mobile agent systems and based on X.509 attribute certificate,presents a privilege management scheme for mobile agent which can offer a universal service for authorization and access control safely and agilely.

Keywords:mobile agent;X.509;attribute certificate

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现

Linux:

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植

13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计

3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法