Linux 下以太网的 IPv6 隧道技术的实现

施晓煌，郭 洪，林常俊，林晓东，黄金填

（福州大学数学与计算机科学学院，福建 福州 350002）

摘要：IPv6 是一种新的互联网协议的规范，由于与 IPv4 不直接兼容，限制了其应用和发展. 为此，提出一种在 Linux 下网络层隧道技术的配置与实现方案，该方案的实现使得各个 IPv6 的信息孤岛能通过 IPv4 网络进行通信，解决了 IPv6 到 IPv4 的转换问题.

关键词：Linux；IPv6；IPv4；隧道；以太网

中图分类号：TP393.03　　**文献标识码**：A

Realization of Linux - based IPv6 tunnel on ethernet

SHI Xiao - huang, GUO Hong, LIN Chang - jun, LIN Xiao - dong, HUANG Jin - tian

(College of Mathematics and Computer Science, Fuzhou University, Fuzhou, Fujian 35002, China)

Abstract: IPv6 is a new criterion of the interconnection network protocol. IPv6 and IPv4 are not directly compatible, which limits the application and development of IPv6. The article is about the realization of tunneling which is based on Linux. Tunneling is a schema dedicated to serve for IPv6 host to communicate with IPv6 host during the transition period of the Internet and solves the translation from IPv6 to IPv4.

Keywords: Linux; IPv6; IPv4; tunneling; ethernet

与 IPv4 相比，IPv6 具有许多新的特点[1]，如简化的 IP 报头格式、主机地址自动配置、认证和加密以及较强的移动支持能力等. 要实施 IPv6 网络，必须充分利用现有网络环境构造下一代互联网，以避免过多的投资浪费. IPv4 也因其出色的技术特性在互联网领域获得了巨大的成功，现在的互联网是基于 IPv4 的，不可能将它们在短时间内都过渡到基于 IPv6 的网络. 因此，在相当长的一段时期内，IPv6 网络将和 IPv4 网络共存. 要最终实现 IPv4 向 IPv6 的平滑过渡，首先要处理现有 IPv4 网络和未来 IPv6 网络之间的相互通信问题. 目前实现 IPv4 与 IPv6 之间互相通信的方法有多种，本研究设计并实现了 Linux 下 IPv6 隧道技术.

1　Linux 下隧道应用类型

在 Linux 下的 IPv6 隧道提供了一种利用 IPv4 网络传输 IPv6 分组的方法. 隧道技术主要有以下几种应用[2]：①路由器 - 路由器隧道：用于连接被 IPv4 网隔离的两个 IPv6 网的连接；②主机 - 路由器隧道：该隧道始于主机，终于中间路由器，用于独立的双 IP 主机通过双 IP 路由器与 IPv6 网进行通信；③路由器 - 主机隧道：用于将独立 IPv6 或 IPv4 节点与 IPv6 网络隔离，以主机为隧道终点；④主机 - 主机隧道：用于将相互独立的 IPv6/ IPv4 节点通过 IPv4 网相互通信，此时 2 个双 IP 节点作为隧道的端节点通过 IPv4 网进行通信.

2　隧道技术的实现

2.1 IPv6 隧道技术通信模型

如图 1 所示，该模型是利用隧道技术进行通信的一个重要的基本模型. 由于该实现方案基于主机到主机隧道技术，该隧道技术是将 IPv6 分组传到主机，可以用 IPv6 分组的信息获得终点地址.

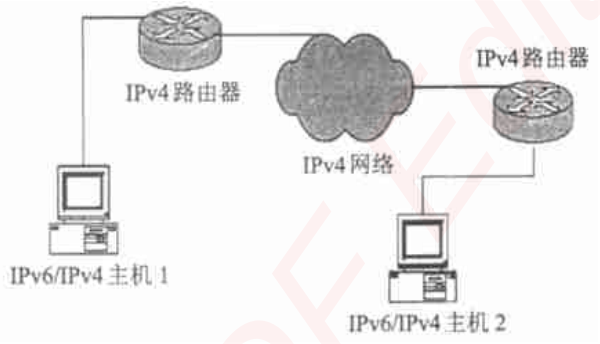

图 1 IPv6 隧道技术通信模型

Fig. 1 The IPv6 tunnel communication module

2.2 Lunix 内核模块编程

Lunix 下的模块将具体的一部分功能块隐藏在抽象的接口背后. 使用模块的最大特点是将接口与其实现分离开来，保证一个模块可以在不影响其它模块的情况下进行改变. 这样也将模块之间的依赖关系仅仅限定于接口. 这些模块是内核的一部分，经编译成独立的目标代码文件后，可在需要时加载.

进行隧道设计必须使用模块编程，一个 Linux 内核模块至少包含 2 个函数：init-module 和 cleanup-module，分别在加载和卸载该模块时被调用. 因此，init-module 在内核中注册某一句柄或者用其代码替换内核中原来的函数. 而 cleanup-module 则作相反的工作以保证模块被安全地卸载. Linux 以 root 身份登陆后，提供了 insmod 加载核心模块与 rmmod 卸载核心模块两个命令，所以可用 insmodipv6 来加载 IPv6 模块.

2.3 编译支持 IPv6 模块

要让 Linux 操作系统加载 IPv6 模块，就要安装 IPv6 协议栈，Redhat8.0 的内核为 2.4.18 版本，可支持 IPv6，但是安装缺省不支持. 由于协议栈在操作系统中是处于核心地位的，必须重新编译新的内核才能安装上新的协议栈.

1) 以 root 身份登陆，进入源码所在的目录：cd/usr/src/linux.

2) 运行 make clean，清除一些可能过期的中间代码.

3) 配置内核选项：make menuconfig 或者 make xmenuconfig，运行 make menuconfig 后，将下面的支持 IPv6 的选项选上，其他内核选项根据系统的具体情况作出符合系统的选择.

```
Code maturity level options
    Prompt for development and/or incomplete code/drivers   yes
Networking options
    Packet socket                                           yes
    Unix domain sockets                                     yes
    TCP/IP networking                                       yes
    The IPv6 protocol                                       yes
    IPv6: enable EUI-64 token format                        yes
    IPv6: disable provider based address                    yes
File systems
    /proc filesystem support                                yes
```

4) 运行 make dep；make clean；make bzImage.

5) 没有错误，编译成功了支持 IPv6 协议的内核. 将该内核拷贝到 Linux 的启动目录下.

6) 编译 etc/lilo.conf 使新的内核成为 root 的选择.

7) 重启系统：reboot. 重启系统后，会发现有支持 IPv6 的内核信息出现，该主机成为安装并支持 IPv6 协议的 Linux 主机.

2.4 IPv6 隧道的搭建

建立好支持 IPv6 和 IPv4 双协议的 Linux 主机后，按照实际 IPv4 网络情况，配置好网络接口的 IPv4 地址. 之后，要分别为主机 1 和主机 2 配置隧道，两隧道的端地址分别为 Host1 和 Host2 的 IPv4 地址. 在确定好支持 IPv6 的内核和建立隧道两端的地址后，可利用 iptunnel 和 ifconfig 来配置 tunnel.

1) 建立 Tunnel - sit1:

iptunnel add sit1 mode sit remote < remote IPv4 > local < local IPv4 > (其中 sit1 中的 1 表示隧道 1，如果是隧道 2 就可以选择如 sit2...)

if config sit1 up

用 iptunnel 的命令来添加隧道，其中 sit1 是定义该隧道的规范名称，并在后面加上两台主机的 IPv4 的地址(本地与远程)，就在两主机之间搭建起一条隧道，通过用 ipconfig sit1 up 将整个隧道激活，经激活该隧道并利用该 sit1 隧道对网络传输中的数据进行挂接，使主机开始以隧道互相传送信息和数据.

2) 配置 sit1 的 IPv6 地址:

ifconfig sit1 inet6 add < local IPv6 > /128

route - A inet6 add < remote IPv6 > /128 sit1

此时整个通信通道已经配置好了，这样隧道就连通了. 隧道 1(sit1)为手动隧道(Configured Tunneling)，在手动配置了 sit1 之后，跨越了 IPv4 网络的 IPv6 主机之间就能进行通信.

3 利用 sit1 对数据进行封装和解封

如图 2 所示，隧道技术将 IPv6 的分组封装到 IPv4 的分组中. 封装后的 IPv4 分组源地址和目的地址分别是隧道入口和出口的 IPv4 地址，再将报头的"协议"域设置为 41，表明这个分组的负载是一个 IPv6 分组，然后在隧道的出口处将 IPv6 分组取出转发给目的站点.

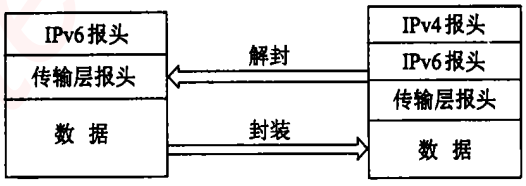

图 2 隧道数据的封装与解封

Fig. 2 Encapsulation and unencapsulation of tunnel's packets

3.1 用 sit1 对数据的封装

放置一个新的 IPv6 基本报头到原始分组之前，以及放置可选的 IPv6 扩展报头，此处的原始分组相对应于传输层中的分组. 这些放置的所有报头总称为隧道 IPv6 报头，封装在 IPv6 隧道的入口完成，然后原始分组被转发到隧道所代表的虚拟链路去. 原始分组在转发过程中处理，其处理是依照该分组协议的转发规则进行的. 在封装过程中，隧道 IPv6 报头的信源字段由隧道入口节点的 IPv6 地址填充，信宿字段由隧道出口节点的 IPv6 地址填充[3]. 因此，封装产生的隧道分组是向隧道出口节点发送的. 这一过程如图 3 所示.

3.2 用 sit1 对数据的解封

解封过程实际上就是封装过程的逆过程. 这一过程可以用图 4 表示. 当隧道出口节点接到一个投送到其 IPv6 地址的 IPv6 分组时，该节点的 IPv6 协议层将处理隧道报头. 为隧道分组解除封装的隧道出口节点和最终收到产生的原始分组的信宿节点可以是用同一个节点.

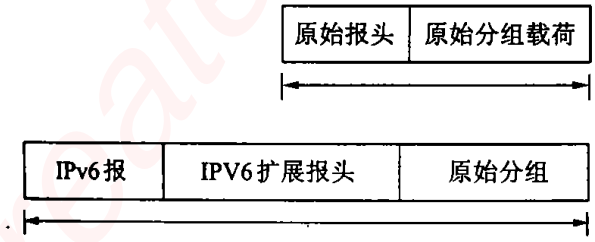

图 3 封装一个分组

Fig. 3 Encapsulation a packet

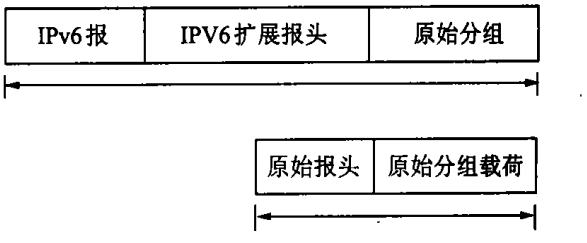

图 4 解封一个分组

Fig. 4 Capsulation a packet

4 利用 sit1 隧道传送数据中涉及的安全问题

sit1 隧道利用安全 Shell，即 SecureShell(简写为 SSH)进行网络的安全访问. 一般的 Unix 系统、Linux 系统、FreeBSD 系统都附带有支持 SSH 的应用程序包. SSH 协议是建立在应用层和传输层基础上的安全协议，因此数据在传输过程中可以利用 SSH 协议在应用层和传输层上进行加密、数字认证，然后再通过网络层将安全的数据进行封装，因此在整个传输过程中数据可以保证其安全性.

本隧道实现方案在 2 台支持 IPv6 的 Linux 主机上测试通过. 该方案是在网络层对传输过程中的数据包进行报头封装和解封，以实现 IPv6 数据能在 IPv4 网络间传送，可用于支持 IPv6 的 Linux 局域网之间互相通信和共享资源.

参考文献:

[1] 伍海桑. IPv6 原理与实践[M]. 北京: 人民邮电出版社, 2000.

[2] 李津生, 洪佩琳. 下一代 Internet 网络技术[M]. 北京: 人民邮电出版社, 2001.

[3] Hinden R, Deering S. IP version 6 addressing architecture[EB/OL]. Http://www.faqs.org/rfcs/rfc2373.html, 1998-07.

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
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现

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
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发

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
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器

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
21. DCOM 协议在网络冗余环境下的应用

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
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计

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
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植

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
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现

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
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现