基于 Davinci 平台的 SD 卡读写优化

梁正和，秦超，陆国强

摘　要：Linux2.6 内核里集成了 Davinci 平台的 MMC/SD 卡驱动程序，但是在实际使用的过程中发现，应用程序对 SD 卡进行读写操作时，处理读写请求的的 mmcqd 进程会造成很高的 CPU 峰值，极容易导致系统异常。基于此点，进一步研究 linux2.6.18 里 Davinci 平台的 MMC/SD 卡驱动程序，解决了 CPU 占用率过高的问题，并大大提高了其传输速度。

关键词：Linux；SD 卡；驱动程序；Davinci

中图分类号：TP202　　**文献标志码**：A

SD Card Reading and Writing Optimization Based on Davinci Platform

Liang Zhenghe[1], Qin Chao[1], Lu Guoqiang[2]

(1.Hohai University, Nanjing211100, China; 2. Nanzi Information Technology, Nanjing 211100, China)

Abstract: The kernel of Linux 2.6 has integrated the Davinci MMC/SD card driver, but in the practical use of the driver, a process named mmcpd will cause a high CPU peak and result abnormal of the system .Based on the point, after intensive study of the Linux 2.6 kernel, the paper solves the problem of high CPU occupancy rate and greatly improves the transmission speed.

Key words: Linux; SD Card; Driver; Davinci

0 前言

随着嵌入式处理器速度的不断提高以及嵌入式系统功能的不断增加，嵌入式系统越来越多地被应用到网络音视频的处理上，在这种情况下，美国德州仪器公司（TI ）推出了面向嵌入式网络音视频应用的Davinci解决方案。这个解决方案的提出使开发人员摆脱了数字视频的具体处理细节，加快了产品上市进程。Davinci解决方案也成为了国内的热门研究课题。

SD 卡（Security Digital Memory Card）是由日本松下公司、东芝公司和美国SANDISK 公司共同开发研制的全新存储卡产品。SD 卡兼容MMC 卡接口规范，并以其大容量、高性能、高安全性等特点成为了嵌入式环境中存储大容量数据的首选。对SD 卡的操作也是Davinci 解决方案里一个必不可少的组成部分。

1 Davinci 平台的 SD 卡驱动程序

1.1 MMC/SD 卡控制器

MMC/SD 控制器通过MMC/SD 协议与SD 卡进行通信，MMC/SD 卡控制器通过配置可以实现MMC 控制器和SD 卡控制器之间的转换。驱动程序可以通过读写控制器里的寄存器和FIFO ，启动一次与SD 卡的通信。通过设置寄存器，可以发送命令和参数、设置接受的应答的格式、事发后发送/接收数据以及是否产生同步时钟等SD 卡操作所需要的功能。控制器还可以进行DMA 操作，将控制的FIFO 作为DMA 控制器的目标或者源，实现后台的数据传输，从而提高系统效率。

Davinci 的MMC/SD 控制器结构，如图1所示：

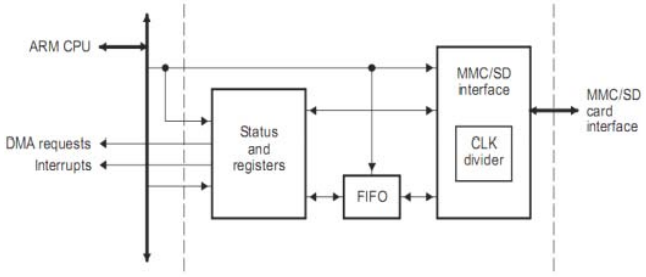

图1

1.2 Davinci 平台的SD 卡驱动程序

Linux2.6 内核支持Davinci 平台下的SD 卡，在drivers/mmc/davinci-mmc.c 里为其提供了具体对应于davinci 平台的SD 卡驱动程序。

SD 卡是块设备，只能以块为单位来进行读写操作。因此SD 卡驱动程序必须将SD 卡实现为块设备。SD 卡驱动程序可以分为4 层，分别为：协议层、块设备驱动层、抽象设备层、具体设备层。

在协议层里，规定了控制器与卡通信的具体命令，以

及每个命令的类型和返回类型等。

在块设备驱动层里，mmc_block.c 首先通过register_blkdev（）向内核注册自己，然后通过driver_register（）来注册对应的驱动。驱动里包括probe 函数、remove 函数、suspend 函数、resume 函数，在其probe 函数里完成了gendisk 结构体的初始化，请求处理函数的设置等块设备驱动程序的核心。

在抽象设备层，mmc.c 里实现了对SD 卡的具体操作，如检测卡的状态、读取SD 卡寄存器。

在具体设备层里，davinci-mmc.c 注册了具体的SD 卡设备和驱动程序。实现具体的中断处理、数据传输、请求函数。

当有用户向SD 卡发出读写命令时，命令会被内核发送到块设备的请求队列里，调用在块设备驱动层里设置好的请求处理函数mmc_request（），mmc_request （）会唤醒mmc_queue_thread 线程，而mmc_queue_thread 会调用mmc_blk_issue_rq （）对请求进行处理，最后请求会被转换为标准协议并提交到mmc_davinci_request （）进行命令和数据的发送与接收，具体过程，如图2所示：

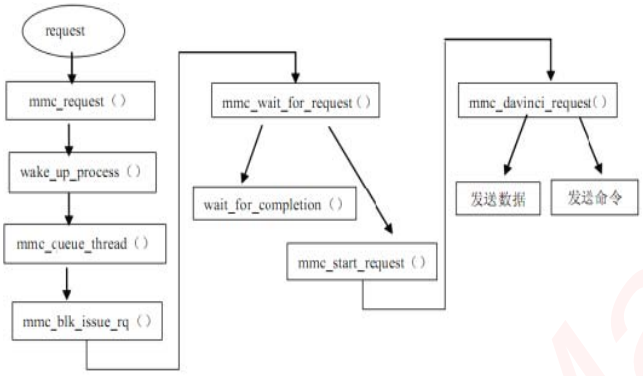

图2

2 SD 卡驱动优化

在Linux2.6 内核里将驱动程序加载上之后，虽然能对SD 卡进行正常的读写操作。

然而，在实际应用中发现，在进行读写操作时，CPU 峰值会达到80%，读写速度仅为2.3MBps 左右，在嵌入式系统资源有限和Davinci平台高实时性要求的情况下，如此高的CPU占用率无疑会影响系统的正常运行。

2.1 降低CPU 峰值

2.1.1 原因分析：

由于SD卡的运行模式会影响其传输速率，因此，首先检查SD卡运行模式：

总线频率：25MHz （正确）

总线位宽：4bit （正确）

信令模式：多块传输（正确）

搬运模式：DMA（正确）

这说明了SD卡运行模式不是造成高CPU峰值的原因。开启内核profile 功能，对IO请求处理过程进行分析，结果如表1所示：

表1

Ticks	Kernel function	Normalized ticks
10094	total	0.0040
4832	default_idle	48.3200
2953	__delay	246.0833
779	mmc_davinci_request	0.4376
257	__copy_from_user	0.2722
139	const_udelay	4.9643
113	mmc_blk_issue_rq	0.1320
77	__make_request	0.0887
64	handle_IRQ_event	0.2540
51	mmc_queue_thread	0.1238

内核profile 功能的原理是每次系统时钟中断时，获取并统计中断前内核PC指针的值。而__delay 占用的系统tick数为2958 远大于IO 请求处理接口函数mmc_queue_thread所占用的tick 数目，即等待请求处理函数对请求进行处理的时间远大于请求本身被处理所需要的时间。

2.1.2 优化

Linux 内核提供了许多延迟方法处理各种延时请求。不同的方法有不同的特点，有些是在延迟任务时挂起处理器，而另外一些则会一直占有CPU。ndelay（） 、udelay（） 、mdelay （）3个函数分别进行纳秒、微妙和毫秒级的延时。这3个函数都是用__udelay （）实现的，而__delay 是忙等待，也就是说在延迟4的过程中不会让出CPU，从而导致循环等待代码占用了过多的CPU 时间。更为理想的方法是使用schedule_timeout （）函数，该方法会让需要延迟执行的任务睡眠到指定的延迟时间耗尽后再重新运行，它由__schedule （）实现的，而__schedule（） 并不是忙等待，它会在等待时让出CPU。

找到davinci-mmc.c 里的轮询代码将循环占用CPU（udelay ）修改为循环睡眠等待（schedule_timeout）。

3.1.3 测试修改后开启内核profile 功能，如表2所示：

表2

Ticks	Kernel function	Normalized ticks
12518	total	0.0050
11342	Default_idle	113.4200
230	mmc_blk_issue_rq	0.2687
139	__copy_from_usr	0.1472
83	__schedule	0.0469
53	__make_request	0.0611
35	handle_IRQ_event	0.1389
32	hrtimer_start	0.1176
30	complete	0.2143
26	add_preempt_count	0.1477
24	mmc_queue_thread	0.0583
24	mmc_davinci_request	0.0122

上述数据说明轮询代码所占用的CPU时间已经大大减少，系统占用系统的瓶颈已经由轮询代码转移到了mmc_blk_issue_rq 函数。

同时，用dd 命令全力进行读写操作，CPU峰值降为20% ，吞吐量为2.1MBps。

2.2 提高读写速度

2.2.1 原因分析

20%的CPU峰值依然很高，而且吞吐量也因此而下降。由于mmc_blk_issue_rq 为SD卡驱动处理IO请求的入口函数，故CPU 峰值过高的原因是IO请求过多或IO请求占用CPU 时间过多。但是由于mmc_davinci_request 占用的系统tick 数很少，所以可以排除单个IO请求占用CPU时间过多的可能。运行blktrace观察IO请求的数量、结果，如表3所示：

表3

179,0	0	20	0.000000000	736	D	W0+16[mmcqd]
...						
179,0	0	583	0.130000000	736	C	W0+16[0]
179,0	0	584	0.130000000	736	D	W0+16[mmcqd]
179,0	0	585	0.130000000	736	C	W0+16[0]
179,0	0	586	0.130000000	736	D	W0+16[mmcqd]
179,0	0	587	0.140000000	736	C	W0+16[0]
179,0	0	588	0.140000000	736	D	W0+16[mmcqd]
179,0	0	589	0.140000000	736	C	W0+16[0]
179,0	0	590	0.140000000	736	D	W0+16[mmcqd]
179,0	0	591	0.140000000	736	C	W0+16[0]

(其中: “179,0”代表SD 卡块设备号, “D”代表issued, ”C” 代表complete, “W” 代表write, “x+y ”代表块设备上从偏移量x 个扇区开始,长度为y个扇区的区域)

Blktrace 的原理为对IO 请求处理路径上的各个关键处理点进行实时统计，上述数据说明了SD 卡驱动每次处理的IO 请求仅为16 个扇区，IO 请求大小仅为8KB（16*512B=8KB），每次传输的数据量过小，导致传输效率低，IO 请求过多，CPU 峰值过高。

SD 卡IO 请求的大小受到DM365 芯片的SD 卡控制器和EDMA 控制器的限制：

SD 卡控制器：每次IO 请求大小最大为64K*512B =32MB 其中64K为最大的数据块个数（MMCNBLK），512B为块大小。

EDMA 控制器：每次IO 请求大小最大为4B*64K*64K*256=4TB 其中4B为传输单元大小（ACNT），64K 分别为传输行大小（BCNT ）和传输块大小（CCNT），256为传输链表大小（LINK）。

因此，理想情况下IO 请求大小最大应该为32M 。

检查EDMA 控制器的参数，发现EDMA 设置传输链表大小为2，传输块大小为128KB ，故可以断定，对传输链表大小设置过小，导致了IO 请求过小。

2.2.2 优化

EDMA3 提供了一种链接的DMA 传输机制，允许整个PaRAM（ParameterRAM） 重新加载，这种机制在ping-pong buffers 、circular buffering 和连续传输里很有效。由于EMDA3 控制器最多支持256 个PaRAM ，故，理论上传输链表大小最大可以为256。

修改驱动程序里EDMA链表的设计，使其支持16个传输块（不采用256 个传输块，因为其他程序也要用到DMA 传输块）。

2.2.3 测试

开启内核profile 功能、数据，如表4所示：

表4

Ticks	Kernel function	Normalized ticks
9968	total	0.0039
8013	default_idle	80.1300
351	__copy_from_user	0.3718
154	mmc_blk_issue_rq	0.1799
89	__schedule	0.0503
78	__make_request	0.0899
54	arm926_dma_clean_range	1.9286
53	__set_page_dirty_nobuffers	0.1452

上述数据说明系统的瓶颈已经由IO请求处理入口函数转移到了write 系统调用的拷贝上。同时，用dd 命令全力进行读写操作，CPU峰值降为10%，吞吐量为6MBps。相比于优化前的80%CPU峰值和2.3MBps 吞吐量，有了很大的提高

3 总结

本文给出了Davinci平台的SD 卡驱动程序的优化方案，经过优化后的SD 卡驱动程序降低了CPU峰值，提高了读写速度，已经完全能够满足davinci平台下的应用。

但在改进的过程中发现，因为DMA 连接数位2，传输块大小设置为128KB，因此IO 请求的大小也应该为256KB，但实际上IO 请求大小却只有8K，说明传输块的大小并没有达到所设置的128K。分析原因后发现由于系统采用4KB 大小内存页存储IO 请求数据，多个连续的内存页作为一个DMA 传输块进行传输，当内存页不连续时，最坏情况下，每个传输块只有一个4KB 内存页，无法达到预期的128K，所以IO请求实际大小只有8K（4KB*2）。未来优化方向为研究如何尽可能的将数据集中在一个传输块内进行传输，进一步提高效率。

参考文献：

[1] （美）Corbet,J 等著；魏永明，耿岳，钟屬毅译. linux 设备驱动程序［M］. 北京：中国电力出版社. 2008

[2] TMS320DM36xDigital Media System-on-Chip(DMSoC) Multimedia Card/Secure Digital Card Controller User's Guide. www.ti.com.cn. 2010

[3] 纪竞舟，付宇卓；嵌入式 linux 下的 MMC/SD 卡的原理及其实现［M］. 上海：上海交通大学芯片 7 与系统研究中心. 2005

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
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