我在盛利亚的 800 天(2009.5.20~2011.7.29)

作者 ： 余祖波(livelylittlefish@gmail.com)
Blog: http://blog.csdn.net/livelylittlefish, http://www.abo321.org

---

Content

零．序

于 2009 年 5 月 20 日加入盛利亚，至今整 800 天。跟儿子同龄，儿子是 2009 年 5 月 15 日来到这个世界的。儿子的到来给我们全家带来了好运，给我也带来了好运，加入盛立亚，就是我的好运的开始。

在即将离开盛立亚之际，对自己在这家公司的工作情况、研究与学习情况进行回顾、总结，写成此文，以此感谢每一个曾经指导、关心、帮助过我的人；同时总结过去、展望未来，希望自己能在未来有更多的积累，更多的进步。

一．感谢篇

在该篇里，要特别感谢这几个人(此处请原谅我用了真实姓名)。

此处，笔者还有其他的希望，希望将来某一天儿子看到这些文字时，能够看到他的爸爸曾经的努力，希望他能够踏踏实实工作、真真实实生活；也希望他学会感恩，感恩所有爱他的人、帮助他的人。也希望将来自己年老看到这篇文字时，仍然能够想象自己在这 800 天里的尝试和努力。于此，真的心满意足。

1．盛立亚

是我们公司的简称，全称是“盛立亚(中国)光网络系统有限公司”(下文用简称)，这是一个不算大的家庭，虽然并没有业界其他大公司、超大公司那么大规模，虽然平台没有他们那么高，但盛立亚是全球最早进行 EPON 产品研发的公司之一。她是 IEEE802.3ad 协议标准的制定者，2010 年又参与了 DPOE 规范的制定。

对于自己的员工，她也一直在尽自己的努力做到最好。对于我个人来说，在这 800 天里，我在盛立亚得到了成长，获得了进步。在这里，你对哪个问题感兴趣，都可以亲自去实验，学习其原理，掌握其真谛。因此，我要感谢盛立亚提供的这个自由的平台，让我可以自由地发挥。

2．李岩波

我们 Team 的 Senior Department Manager。当初就是这个牛人老大把我招进来的，技术是我们盛利亚的 No.1，对多种技术都有自己深刻的理解，例如，C/C++/Java 等都有深入的挖掘和理解，对面向对象、设计模式、分布式同步/并发模式等都有深刻的理解，对通信行业有深入的实践和认识，对通信等中间件软件 ACE/ICE/OpenSAF 等都有深刻的认知，等。尤其擅长大型系统架构、疑难问题的解决，公司有不少疑难问题都是他深入挖掘并解决的。

他为人热忱、做事低调、工作认真、负责，思考问题深入，理解问题全面。另外，还特别会鼓励下属，此处深深地感谢他在平时的工作中将一些有一定挑战性的任务交给我，并给予指导和帮助。从他身上，我学会了很多，感谢上帝让我成为他的 Team Member。

可通过LinkedIn Profile了解他的详细信息。

3．荆华

我习惯称他为“华仔”，实际上这个称呼从大二就开始了。他首先是我的大学同班同学、研究生同班同学，如今是同事。当时也是他推荐了我加入盛立亚。用我老婆的话说“华仔就是你的贵人，他把你推荐到哪里，那里的头就很喜欢你”。是的，华仔是我一生的朋友，是我最珍贵的财富。

他对生活的态度，对工作的积极、认真和负责，对他人的热情和真诚，无一不在影响着周围的同事，无一不在打动着我。在技术上，尤其擅长嵌入式 C 开发(VxWorks 系统或者 Linux 系统)、软件设计和文档写作，对 Linux 开发，尤其是内核协议栈有深入的了解，在嵌入式软件开发有深厚的积累，他在通信领域积累的行业经验、开发经验、处事经验等，足以让他可以成为该行业的行业专家和开发专家。

在此感谢华仔在盛立亚对我的帮助，作为我的 Monitor，他耐心和仔细的讲解让我很快对 EPON 设备、系统有较为清晰的认识。感谢上帝让我们成为同学、朋友和同事。

二. 工作篇

关于 2010 年详细工作总结，可以参考2010 年度总结。此处简单回顾两年多的时间所做的工作。

1. 工作回顾

首先介绍一下公司的产品(我经历/参与的产品及承担的任务)。

(1) SPON3.9：该版本主要是加入新类新的 ONU，如 ONU3350GDP/3350GR。在该项目中我的主要任务是对 2 层协议模块，例如 IGMP、MSTP、VLAN、PIM、AM、SAL、SM 等，加入对新类型 ONU 的支持。实际上，这项任务非常简单，代码的增加量和修改量都很小，但关键是要认真、仔细，才能保证新类型的 ONU 所有的协议模块都支持。该项任务也是在盛立亚开始上手的工作。

(2) SPON3.10：该版本稳定运行时间很长，改动较少，我的主要任务是 IGMP(v1/v2)协议模块维护的工作，包括 Bug fix。

(3) SPON3.11：该版本主要是加入 Link Allocation 的功能，以及新类型的 ONU，如 ONU4100 等，我的主要任务仍然是 IGMP(v1/v2)协议模块维护的工作，包括 Bug fix。

(4) SPON3.12：该产品应该是修改了 LC(Line Card)板卡的架构，更换了其中用到的 NP(Network Processor)等芯片，其主要工作在 Transport team 的开发，2 层协议的开发工作很少。且因 IGMP 模块已经很稳定，基本不会再有 Bug 出现，故该产品经历的很少了。

(5) DePON1.0：2009 年加入盛立亚时该产品已经发布，没有参与该产品的开发。

(6) DePON1.5：该产品的研发，公司投入大量的人力、物力和财力，历时一年(半)吧，最终在 2010 年 9 月末 release。该产品也是我在公司所经历的一个重要产品，且参与度最多的产品。虽然不知道该产品的市场如何、部署如何。在该项目中，主要承担 Multicast 模块开发、QoS(Quality of Service)模块重构、AM(Authentication Module)模块与 Radius Server 通信等开发工作。

(7) DPoE1.0：该产品实际上由 DePON1.5 发展而来，遵循 DPoE 规范(该规范由全球通信领域几家知名公司共同开发、制定，盛立亚是其中之一，且是第一个通过 DPoE1.0 测试的公司)。在该项目中，主要负责 QoS 模块新增 MIB 表(Management Information Base 管理信息表)的支持。

(8) DPoE2.0：该产品相较于 DPoE1.0 有很大的改变，除加入新的功能(主要表现为业务逻辑)外，有些模块，例如 QoS 需要完全重新(全新)设计。在该产品中，我的主要任务就是 QoS 模块的重新设计和实现，包括架构设计，TLV(Type Length Value)子模块、CM(Cable Modem)管理子模块、CMTS(Cable Modem Terminal System)

管理子模块、SNMP Container(Simple Network Management Protocol)子模块、L2VPN(Layer 2 Virtual Private Network)子模块的设计等。

其他的产品例如 SAM 系统(Jave Web 版设备管理/配置系统)没有参与，此处不再介绍。

注 1：EPON(Ethernet Passive Optical Network)：以太无源光网络。OLT(Optical Line Terminal)：光缆终端设备。ONU(Optical Network Unit)：光网络单元。其他的会在用到的时候解释。

注 2：SPON(Standard EPON)系统是我们公司的主打产品，在北美市场部署范围也很广泛，北美 4 大电视运营商是我们的主要客户，系统也稳定运行很长时间。上述 SPON 系统即指 OLT 设备及其上的软件系统。后文如不特别指出，一般指其上的软件系统(该软件属于 VxWorks 系统上的嵌入式软件)。

注 3：DePON(DOCSIS EPON)系统和 DPoE(Deprovision Over EPON)系统实际上包括两部分：OLT 设备及其上的软件系统(Vxworks，C，嵌入式软件)和 DServer 系统(Linux，C++，系统软件)。此处的"OLT 设备及其上的软件系统"与 SPON 实际上是有一定区别的，主要区别在 DePON 系统中定义，此处不赘述。

2．重要任务

(1) DePON1.5 产品设备系统IGMP模块V3 版本开发(vxWorks, C)

主要任务：在 IGMPv1/v2 基础上，加入对 IGMPv3 的支持。(IGMP 属于 OLT 设备上的软件，属于嵌入式开发。)

关于IGMPv3(这是难点)

- 对于 IGMPv3 的以下特点如何处理包？
  两种模式：Include/Exclude Mode
  6 种 Group Type：
  - MODE_IS_INCLUDE/MODE_IS_EXCLUDE
  - CHANGE_TO_INCLUDE_MODE/CHANGE_TO_EXCLUDE_MODE
  - ALLOW_NEW_SOURCES/BLOCK_OLD_SOURCES

  3 种查询方式：General Query，Group-Specific Query, Group-and-Source-Specific Query
- 如何正确地实现IGMPv3 复杂的状态机？(关于状态机可参考协议原文RFC3376)该问题是重中之重。该状态机能否画出？如何理解每一个状态及状态转移？
- 如何设计数据结构以保证 Source/Group 的 timer 的设定或修改？
- 如何保证与IGMPv1/v2 的兼容？(关于v1/v2 可参考RFC2236/RFC1122)
- 如何进行 event 上报(DePON 系统中 DServer)？以便 DServer 系统的 Multicast 模块的处理？
- 如何向单个 LC 板卡发送 ICC(Internal Card Communication)消息？又如何向所有其他 LC 板卡发送 ICC 消息？
- 。。。

对于刚进公司的我来说，这是一个挑战(这里要再次感谢我的 manager 李岩波老大，将这个有一定挑战性的任务交给我)。这些问题都需要搞清楚，才能够写代码，也才能更清楚地理解设备工作原理、系统对数据的处理流程。最初的进展很慢，因为要边学习边 coding。但经过坚持不懈的努力，终于在很短的时间里搞清楚所有问题并写完代码、完成测试。当然，最后的测试发现，该模块的 Bug 真的是很少。

因笔者的主要工作是二层协议的开发，有一些相关问题，如果能有些了解或者掌握，将会更好。此处可参考笔者总结或

者翻译整理的一些文章，或者一些参考资料，如下。

关于协议

- EPON 系统架构，原理(可参考《无源光网络原理》)
- IGMP 协议原理(可参考如下文章)
  IGMP技术总结（1）
  IGMP技术总结（2）
- MPCP 协议原理(可参考如下文章)
  IEEE802.3ah协议学习（1）——多点MAC控制简介
  IEEE802.3ah协议学习（2）——多点MAC控制操作
  IEEE802.3ah协议学习（3）——MPCP多点控制协议
- TCP/IP 协议架构、原理(可参考《TCP/IP 详解》)
- RPC协议(可参考RFC1050, RFC1057)等

对于 IGMP 模块来说，因系统对 IGMP 包处理的特殊性，需要重点掌握其原理。已有的系统，要重点掌握 IGMPv2 的状态机处理流程。具体来讲，需要彻底搞清楚如下问题。

关于IGMPv2

- IGMP 模块如何架构的？又有哪些子模块？这些子模块间如何通信？典型的通信流程是怎样的？
- SPON 系统和 DePON 系统中的 IGMP 模块各有什么不同？上层业务有多少差别？上层业务的差别导致 IGMP 模块实现上有区别吗？
- 从 DePON1.0 到 DePON1.5 又增加或修改了哪些上层业务？这些变化导致 IGMP 模块中有各有什么不同的处理？
- 对于 IGMP Snooping 和 IGMP proxy 两种模式，IGMP 模块本身是如何设计这两种模式的？又是如何处理的？处理流程如何描述？即 IGMP 模块接收到这些报文后都要做什么？(这一点需要详细了解 IGMP 协议的状态机，需要对每一步都有较好的理解和掌握)，亦即 IGMP 的状态机是如何实现的？能否画出 v2 的状态机？如何理解状态转移？
- 对于两种不同的模式，IGMP 协议报文在系统里如何流动？Query、Report、Leave 报文各如何？TK(Teknovus)芯片、NP 芯片如何处理，处理又有什么动作？
- 对于这些报文，IGMP 模块本身如何处理？如何触发的？两种不同模式的 Group，IGMP 模块有哪些相关数据结构，是如何组织这些数据的？
- IGMP 模块的定时器有哪些？分别做什么？区别在哪里？如何调试这些定时器？
- 是否考虑过处理过程中的优化？如果有，如何优化的？能不能更好的优化？如果没有，又该怎样优化？
- 如何实现 CLI 对 IGMP 模块相关数据的查询？查询过程中，是否有板间通信发生？有哪些消息流动？数据是如何流动的？有哪些相关操作？
- 如何通过 CLI 对 IGMP 模块进行配置？如何处理这些配置命令？如何同 CLI 进行交互？
- IGMP 模块与其他模块，如 SAL(Service Agreement Level)，VLM(Virtual LAN Management)等，有无消息交互？有，交互流程如何？能否用时序图表示？
- IGMP 模块正常工作的基本配置又是怎样的？如何快速地链接并测试 IGMP 模块？
- IGMP 模块如何调试？有哪些调试命令？
- ...

关于设备系统，要想很好的理解其工作原理和对数据的处理流程，仍有不少问题需要搞清楚(作为一个软件开发者，想要深入理解某个系统的工作原理，需要深入地思考问题的本质。要能解决工作中的问题，更要学会问问题。我想，这也应该是成为一个优秀的开发者需要具备的基本特质吧。)，如下。

关于系统架构

- 系统如何架构？系统又如何分层？层与层之间如何通信？
- 系统如何启动？我们的系统采用的是分阶段启动，那么分几个阶段？每个阶段又 spawn 哪些任务？SCC(System Control Card)卡和 LC 卡有什么区别？
- 系统有哪些模块？各个模块都有哪些任务(这些任务跑在 VxWorks 系统上)？这些模块的任务是如何定义的？
- 如何查看这些任务？这些任务使用的 CPU、栈大小、优先级、任务入口等信息如何知道？与 LC 上任务相关的这些信息是如何组织的？SCC 卡和 LC 卡有什么区别？
- 模块启动后如何初始化？初始化动作由谁触发？然后又如何 restore？restoration 动作又是如何触发的？

关于代码组织

- 系统代码如何组织？如何自动构建(当然是通过 makefile 工程)？那么 makefile 工程如何组织的？
- 如果想对某些模块进行改写，又如何修改相应的 makefile？比如通过 gcc 得到一些调试信息、头文件依赖信息(header file dependency)等信息？
- 各模块代码如何组织？有没有编码规范？要加入新的模块，如何做？

关于平台支持

- 系统支持的 LC 类型有哪些？有什么区别？系统支持的 ONU 类新有哪些？有什么区别？
- 系统有没有定时器？有，那么定时器如何工作？有哪些类型的定时器，如何使用这些定时器？
- 板间如何通信？系统的 Node 有哪些类型？如何组织？如何识别？ICC(Internal Card Communication)模块是如何设计的？其基本原理是什么？如何实现的？又如何使用？
- 系统在什么 OS 上运行？系统与 OS 间的接口怎样？如何交互？通过系统调用，还是其他？平台如何封装？

关于开发、编译、测试

- 系统开发环境如何？编译环境如何搭建？
- 代码如何管理？bug 如何管理？开发流程如何？
- 如何搭建简单的测试环境？测试系统如何进行基本的配置？
- 如何上传 sf(vxworks 系统上的镜像文件)到 LC 的 flash？有哪些方法(3 中方法)？
- LC 启动的 BootLine 在哪里配置？如何配置？有无命令支持？

关于调试

- 系统如何调试？如何跟踪？每个模块有没有帮助命令？
- 系统的 log 信息如何查看？如何分析？每个模块的调试开关？

关于ONU上线、DDL(Device Dependency Layer)、包处理

- ONU 上线过程是怎样的？如何 discovery links？谁负责 discovery links？Discovery 过程遵循什么协议？
- LC 上的 TK 芯片如何识别协议包类型？识别协议包后，TKD(TK 芯片的 driver)如何将该包 check out 给相应的包处理程序？即这些 callback 如何触发？这些 callback 又如何注册？注册在何处？如何组织？
- TK 又如何处理这些包(当然，只需要对 IGMP 协议包的处理很清楚就可以了，也没有那么多精力搞清楚所有的事情)？处理完这些包之后如何交给 NP(Network Processor)？
- TKD 有哪些相关任务？每个任务入口在哪里？任务中主要流程是什么？
- TK 有哪些消息类型？向 TK 芯片中写入规则，由谁触发？写入流程是什么？如何写入？有没有什么第三方工具可以直接查询？对与 IGMP 模块来说，其相关的规则有哪些？规则是如何定义的？规则的格式是什么？有无调试命令查看这些规则？如何跟踪规则写入过程？
- NP 如何识别从 TK 上来的包？NPD(NP 芯片 driver)如何将包 check out 给相应的模块进行处理(如 IGMP 模

块)？即上层协议 callback 如何触发？这些 callback 在 NPD 中如何注册？如何组织？

- NPD 有哪些相关任务？每个任务入口在哪里？任务中主要流程是什么？

关于其他模块

- SAL、VLM、ICC、TMM(Timer Management)等重要模块如何设计的？其基本工作原理是什么？
- 这些模块对 ONU 上线、CLI 命令等事件，其基本处理流程是什么？如遇到相关问题，能否做一些简单分析？

这些问题，有些属于系统架构问题，有些属于协议处理问题，有些属于平台相关问题，有些属于 transport 问题，甚至有些属于芯片级的问题。如果对这些基本问题都有一个较为清醒的认识和理解，对你后续的工作一定会有很好的帮助。对于芯片级的问题，或多或少地了解一些，也会对你深入理解系统有很多帮助。

上述问题都是笔者在完成工作任务之余的研究、思考、实践所得。因为，这是笔者对自己刚接触该领域的基本要求。

因此，在此建议公司同仁，对新招进来的员工，可以根据其个人实际情况，先试着培训一下新员工对系统的认识，因为有些新员工以前并不一定接触过这样的设备，对系统的原理可能没有多少概念。对于公司现在做的 AMN6400 的设备，也可以有这样的培训，这样会让新员工上手更快，理解更深刻。当然，这也要求新员工对设备有很好的理解，对软件、软件开发有很好的悟性。

所有这些问题你如果都搞清楚了，那么恭喜你，你对系统本身的工作原理应该有一个比较好的理解了，至于业务、协议等的理解，每种业务的实现都不尽相同，需要专门研究相应的官方文档，搞清楚一个协议的开发，例如 IGMP，其他的也能举一反三、融会贯通。

上述几乎所有问题，笔者都有比较深入的理解，并做了大量相关实验，写了很多相关文档。笔者希望，此处提出的问题能对他人起到抛砖引玉的作用。

(2) DePON1.5 产品DServer系统Multicast模块开发(Linux, C++)

主要任务：DServer Multicast开发、维护。该模块基于DOCSIS (Data Over Cable Service Interface Specifications)3.0 规范，支持IP Multicast Join Authorization，支持CLI、RPC、LDB、SNMP(MIB)等。

关于Multicast模块

- Multicast 模块如何设计？又有哪些子模块？这些子模块间如何交互？怎样架构？
- 该模块有哪些对象？对象间有什么关系？如何用 OO 思想来设计？
- Multicast 模块如何启动？如何关闭？由谁触发？怎样触发的？模块启动后有无 restoration？restoration 过程有哪些操作？
- 如何处理 CLI(Command Line Interface)的命令？每个命令的相关操作流程是什么？
- 如何处理 SNMP(Simple Network Management Protocol)的命令？处理流程怎样？是否能清晰描述？
- 如何支持第三方 MIB 工具，如 MibBrowser，对 MIB Table 的读/写？
- 是否需要同设备 OLT/ONU 交互？如何交互？交互过程是否能够清楚地描述？
- 有哪些 RPC 消息？如何发送 RPC 消息？Stub 代码如何编写？Skeleton 代码又如何编写？
- stub 代码是否能自动生成？如果能，如何定义规则，如何自动？
  (关于自动生成的问题，笔者实现了自动生成的代码，并对 Multicast 模块和 BVM 模块定义了相应的规则，且验证了其自动生成。)
- 对 CM 上线，该模块的处理流程是什么？即 CM 如何做 restoration？

- 如何处理设备系统中 IGMP 模块上报的 event？有哪些 event？
- 如何保证 LDB(Local Database)、MIB(Management Information Base)、GDB(Global Database)、内部数据的一致性？
- 如何有效地同步 LDB、设备 GDB、MIB 之间的数据？何时同步？如何触发同步过程？
- 在该模块中，对哪些数据的访问可能会引起竞争条件？如何保护？
- 该模块如何调试？有哪些调试方法？

同样地，与设备(软件)系统一样，关于 DServer 系统，也有类似的问题需要搞清楚，才能对 DePON 整个系统(包括设备和 DServer)的工作原理、数据处理流程、软件架构等有深入的理解。此处，只列出关于系统架构的补充问题，其他相关问题省略，可参考前文设备系统的相关问题。

关于系统架构(补充问题)

- 系统如何架构？该架构如何分层？层与层之间如何通信？
- 模块间有没有依赖？依赖程度如何？如何分析依赖程度？如何尽可能地减少模块间的依赖？
- 常用的代码静态分析工具有哪些？都是代码什么方面的静态分析？
- DServer 系统和设备之间如何通信？RPC 模块的线程如何处理消息？如何触发 RPC 线程？
- RPC 消息处理流程是怎样的？RPC 模块是多线程吗？如何设计？

关于代码组织

关于平台支持

关于开发、编译、测试

关于调试

关于CM上线

- CM 上线流程？有哪些模块参与？CMM(Cable Modem Management)模块主要做什么？该模块有哪些任务？任务线程如何启动？有哪些消息交互？
- CM 上线过程，其配置文件(Configuration File)解析及根据配置将 port 装入 vlan 的流程？该流程有哪些模块参与？能否画出时序图？这期间与 OLT 有哪些交互？OLT 上又有哪些模块被触发？被触发的模块又有哪些动作？

关于事件上报

- 动态转发表 DFDB(Dynamic Forwarding Database)事件上报流程？该流程有哪些模块参与？能否画出时序图？这期间有哪些消息传递？对消息又如何处理？

关于CMTS/CM的interface

- 系统中 CMTS 和 CM 各有哪些类型的 Interface(CM GE/PON/FE, CMTS PON/GE)？
- 如何初始化这些接口？何时初始化？这些接口信息保存在哪儿？如何管理这些接口？如何设计的？能否改进(重构)？

关于其他模块

- DServer 系统的其他模块，如 RPC(Remote Procedure Call)、TMM(Timer Management)、SC(System Controller)、BVM(Bridge VLAN Management)、CMM(Cable Modem Management)、QoS(Quality of Service)等重要模块如何设计的？其基本工作原理是什么？
- 这些模块对 CM 上线、CLI 命令等事件，其基本处理流程是怎样的？消息如何传递？模块间如何交互？如遇到相关问题，能否做一些简单分析和调试？
- 关于 DServer 的常用工具类，比如 Lock，MD5 等小模块，如何设计？基本原理如何？如何使用？
- 如何使用 ACE 的主动任务 ACE_Task 来管理 DServer 系统上的某些子任务？如何使用 ACE_Thread 来管理 DServer 上的线程？与 POSIX 提供的 pthread 库有什么区别？究竟该如何选择？

关于 DServer 系统，笔者同样也做了大量相关实验，写了很多相关文档，对这些问题也有比较深入的理解。笔者同样希望，此处提出的问题能对他人起到抛砖引玉的作用。

(3) ACE系统研究(Linux, C++)

主要任务：

- AM(Authentication Module)模块和 Radius Server 间通信子任务开发；
- 研究 ACE(Adaptive Communication Environment)系统，并将 ACE_Reactor 框架应用到 AM 模块中；
- 将 ACE_Task/ACE_Event_Handler 框架应用到 MCAST、BVM 模块；
- 研究 Glibc, msvccrt, Linux kernel 关于 ACE_Reactor/ACE_Event_Handler 的 handle_events, select 原理；
- 编写 ACE demo 代码，测试 ACE_Reactor, handle_events, select 等，并研究 Expect, TCL，编写自动测试脚本以自动测试所有 demos。

注：后两个任务是笔者的学习计划。

ACE系统是一个优秀的、高性能、跨平台、开源的通信中间件系统，是用C++编写的面向对象的工具开发包，它实现了通信软件的基本设计模式。它面向在Unix/Win32 平台上开发高性能通信服务的开发人员，简化了面向对象的网络应用程序和服务的开发，这些程序和服务用到了进程间通信，事件分离，直接动态链接和并发机制等。ACE通过在运行时动态链接服务到应用程序和在一个或多个进程或线程中执行这些服务自动完成系统配置和重新配置。ACE的分层设计思想也非常值得借鉴。

关于ACE涉及的设计模式可参考POSA(Pattern Oriented Software Architecture)系列书籍，想学习就Google一下吧。笔者在任务中使用的相关部分可参考第 2 卷第 3 章和第 5 章。其他ACE资料，如<C++网络编程,卷 1-运用ACE和模式消除复杂性>、<C++ 网络编程,卷 2-基于ACE和框架的系统化复用>、<ACE程序员指南>等，都是很好的参考资料。

除完成工作任务，在工作之余，笔者重点研究了以下内容，希望能从中学习。

关于ACE Overview

- ACE 是什么？做什么？有哪些应用？
- ACE 系统代码如何组织的？如何编译？如何安装(Linux 平台和 win32 平台)？如何调试？如何跟踪？

关于ACE系统架构

- ACE 系统架构是怎样的？如何分层？如何做到跨平台？
- ACE 系统日志子系统如何设计的？如何在客户自己编写的代码中使用日志记录，如何分析 ACE 系统的动作流程？

关于主动任务ACE_Task

- ACE 系统的任务 ACE_Task 有什么特点？如何设计的？如何实现的？又如何使用？ACE_Task::putq()有什么特点？队列满了怎么办？
- 跟任务相关的 ACE_Message, ACE_Message_Block, ACE_Data_Block 等有何特点？如何设计并实现的？

关于反应器ACE_Reactor

- ACE 系统的反应器 ACE_Reactor 有什么特点？如何设计的？如何实现的？其设计模式是什么？
- 如何识别事件？又如何对事件进行分发？其 handle_events 流程是怎样的？
- 在 Linux 平台上的 ACE_Select_Reactor 是如何设计的？又是如何实现的？如何使用该版本的 Reactor？在 Linux 平台上，其 handle_events 流程又是怎样的？和 Linux Kernel 又有哪些交互？如何交互的？
- 在 Win32 平台上，又是怎样的？

关于ACE的设计模式

- ACE 系统的设计模式有哪些？Half Sync/Half Async(半同步/半异步模式)的原理？在 ACE 中如何应用的？Leader/Follower(领导者/追随者)模式的原理？在 ACE 中如何应用的？
- ACE 的 OS_Wrapper 层如何封装的？使用什么设计模式？

其他问题

- ACE 系统对 TCP、UDP 如何封装的？如何设计并实现的？如何封装网络的基本功能？
- ACE_OS 中有哪些主要功能？ACE 如何实现 Lock、Thread 等 OS 的基本功能？
- ACE_OS::send 是如何送出消息的？又将消息送到了哪里？(ACE_Pipe 结构的管道中)
- ......

关于ACE_Reactor的细节问题，可以参考笔者的2010 年度总结中的叙述。

对于 ACE_Reactor，要想有深刻的理解，在 Linux 平台上，还需要搞清楚 Linux kernel 中相关的一些问题。

- 在 Linux 平台上如何识别事件，即如何实现？(select 系统调用)Linux 上 select 如何实现的？从 ACE 所在的 User Space 如何到 Glibc、又如何到 Kernel Space 实现睡眠(没有事件发生时)？Linux kernel 中线程如何睡眠？
- 什么是系统调用？Linux kernel 系统调用是如何实现的？内核 2.6.18 之前和之后实现上有什么差别？系统调用是如何进入 Linux Kernel 的？
- Linux Kernel 睡眠是如何实现的？futex (Fast Userspace Mutex)在 Linux Kernel 中是如何实现的？有哪些 papers 可以参考？

关于 ACE_Task::putq()结合 Linux Kernel 的 futex 的分析，可参考笔者的一篇文章Linux 平台调用 ACE_Task::putq 分析（timeout=0，队列满），或者下载该文档Analysis of ACE_Task-putq with timeout=0 when queue is full on Linux platform。

当然，有几个问题笔者现在还不能回答，例如设计模式有哪些，这需要对 ACE 系统有比较详尽的了解。关于 ACE 的学习，笔者也做了大量实验，分析了大量的 ACE 代码，并写了很多相关文档，笔者希望以后能抽时间重新整理该专题，并发布到 blog 上。

(4) DePON1.5 产品DServer系统QoS模块重构(Linux, C++)

主要任务：

- 深入理解 QoS 模块功能、架构、设计思想、程序结构、数据结构等，列出所有的不合理，对其全面重构；
- 重构包括删除重复代码，抽取函数，重新设计对象，字段转移，重新设计数据结构，重新设计程序结构，数据优化，性能优化等，数据结构和程序结构等采用面向对象思想设计；例如，
  重新设计 MIB Container, Post Configuration, RPC Generator & Sender 等子模块的数据结构和程序结构；
  采用多态处理 CPU-less 和 CPU-based 的 CM，non-L2VPN 和 L2VPN(L2VPN 又有多个类型)，upstream 和 downstream；
  将 Boost::multi_index_container 应用到 MIB 数据的存储中，简化了数据结构和代码编写，并提高了程序效率；
  重新对待访问数据的竞争条件，重新设计访问同步；

要做好 QoS 重构，务必非常熟悉 QoS 模块代码本身，对其功能、架构、设计思想、程序结构、数据结构等必须非常熟

悉。关于重构的其他问题如下，也务必非常清楚。

- QoS 模块代码到底有哪些不合理的地方？(深刻理解该问题是进行重构的前提)能否列举出来？(要求列举力求全面、客观)
- 针对这些不合理，如何设计你的重构过程？通过哪些步骤进行重构？每个步骤要做哪些重构？
- 如何保证重构的正确性？合理性？重构过程中如何测试？
- 引入哪些设计模式？如何设计、实现？
- 如何重新架构 QoS 模块？如何重新设计 QoS 的子模块，如 TLV, CM/CMTS Service, CM/CMTS Object Management 等；
- 重新设计的子模块间如何交互信息？对数据的访问有无竞争条件？如何保护？

(5) DPoE2.0 系统QoS模块设计、开发(Linux, C++)

主要任务：

- 与另一个同事合作对 QOS 模块采用面向对象的思想进行完全的重新设计、开发；
- 撰写详细设计文档。

新的设计采用层次结构，将 QoS 模块所需的数据按照处理过程分层，并设计子模块，且所有处理完全面向对象，应用了一些设计模式，如工厂模式、strategy 模式、组合模式等。保存数据的链表采用模板设计，并加入 Iterator(迭代器)将对链表的访问接口统一。设计的同时，考虑扩展性、稳定性的同时，考虑模块的可测试性。

很多问题在 QoS 的重构中已经考虑，但要重新设计且支持新功能、新业务的情况下，仍需考虑以下几个问题。

关于设计

- QoS 要处理哪些数据？处理的过程是怎样的？如何划分、设计子模块？数据在 QoS 模块内如何流动？每一个子模块对数据又有怎样的处理？如何重新架构 QoS 模块？
- 如何重新设计 TLV parsing？CLI？TLV CM Objects？CM Objects？CMTS Objects？SNMP Objects？MIB Container？CM L2VPN？等。以及这些对象的管理器 Manager？如 CM Objects Manager, CMTS Objects Manager, SNMP Objects Manager 等。采用什么设计模式？如何正确地将面向过程的设计改为面向对象的设计？
- 他们的数据结构和程序结构如何？如何用模板设计带索引的链表结构？如何为该链表模板提供 Iterator？
- 如何考虑代码的可测试性？

(6) 重度 bug 解决

关于笔者负责的 IGMP 模块(不同产品的 IGMP 模块，如 SPON 系列，DePON 系列)、Multicast 模块，修正 bug 大概 30 多个，其中的疑难 bug，笔者均写有详细的重现、分析、解决文档。其他模块相关的重度 bug 如下。

- CFM(Connectivity Fault Management)模块栈溢出导致 LC 不停重启。
- SCM 模块 ONU 在 restoration 阶段由于没有清除内存导致的内存泄漏，从而导致 LC 重启，然后死掉。
- DFDB 事件上报(MAC override 模式)时，事件太多导致 ICC 消息发送缓冲区溢出，从而导致 RPC 消息丢失。
- QoS 模块插入 map 结构，因编译单元 include 不同的 lock 定义，导致同一个对象在不同编译单元有不同的内存结构，从而导致系统 core dump
- AM 模块与 Radius Server 通信线程调度问题导致在 AM 与 Radius Server 之间没有交互时 AM 的任务仍然运行等待事件发生，这个等待多少让人感觉任务的不自然。建议修改为需要 AM 与 Radius Server 的通信任务时，

再打开，如交互完成，该通信任务便退出，从而更好的利用资源。

以上bug，例如CFM模块的bug，当时并无可考经验或文档，笔者所能得到的信息，只有call stack、每一步调用的返回地址、stack dump(16 进制数据)、相关寄存器的值，笔者结合源代码、PPC汇编代码和寄存器当前值研究、分析stack dump内容，最终发现root cause是该模块的任务栈溢出导致call stack的最后一个调用的返回地址被改写，从而导致系统的异常行为。关于该问题，笔者写了详细的说明文档<Analysis of OLT3540 & OLCDV auto-reboot because of CFM stack overflow during Y1731 test>。实际上，如果一开始就能猜到是栈溢出，修改栈的大小，就能一下子解决问题，但笔者在该过程中积累了stack dump数据分析，汇编代码分析等经验。关于这些问题，可以参考的笔者的文章有过程调用与栈帧，《深入理解计算机系统》3.38 题解—缓冲区溢出攻击实例。

关于 DFDB 事件上报的问题，笔者让系统长时间运行并进行了大量的实验，得到大量的统计数据，最终发现是因 ICC 消息发送队列溢出，从而导致 RPC 消息丢失。

其他 bug 不再赘述。笔者从这些 bug 中，进一步理解系统的工作原理、数据流程和业务等。

(7) 其他小任务

这些任务均在 1~3 天的时间完成。

如 DServer 系统加密算法合并，如 MD5 算法、HMAC 算法，RPC Stub 代码自动生成，读写锁实验等问题。DPoE1.0 系统 QOS 模块新增 MIB 表的支持，主要涉及 QOS 支持的新增 MIB table 代码开发，修改 SNMP 模块相关代码等。

3. 研究、培训任务

(1) OpenSAF 架构研究/workshop

主要任务：对OpenSAF系统进行研究，主要从上层架构、设计原理、如何提供服务等方面考虑。做好PPT后在公司做workshop，做一些架构等方面的介绍。

关于该研究任务，笔者主要考虑以下问题。

- 服务可用性联盟(SAF, Service Availability Forum)，服务可用性接口(SAI, Service Availability Interface)，应用接口规范(AIS, Application Interface Specification)都是什么？他们有什么关系？
- SAI 如何定义高可用性(High Availability)？服务连续性(Service Continuity)？如何利用平均无故障时间 MTBF(Mean Time Between Fault)、平均维修时间 MTTR(Mean Time Between Repair)计算可用性？
- SAI 如何定义故障(Fault)，错误(Error)，失效(Failure)，冗余(Redundancy)？
- AIS 是什么？AIS 如何架构？AIS 有哪几种类型的服务？
- AMF(Availability Management Framework)是什么？
- 集群服务 CLM(Cluster Membership Service)是什么？有哪些节点？节点间如何通信？
- HPI(Hardware Platform Interface)是什么？HPI 是如何架构的？
- OpenSAF 是什么？有哪些应用？可以在哪些环境下工作？
- OpenSAF 的架构是怎样的？有哪些组件？由谁开发？如何开发？如何组织代码/文档等？
- OpenSAF 如何提供五个 99.999(99.999%)的可用性？又如何提高系统可靠性？

- Node 节点有哪些种类？各有什么特点？Master 和 Cluster 分别负责做什么？
- OPenSAF 提供哪些服务？这些服务之间有何联系？服务之间如何通信？借助第三方？还是自己开发？
- 全局锁服务 LCK(Global Lock Service) 有哪些特点？如何提供锁服务？
- 检查点服务 CKPT(Checkpoint Service)是什么？基本原理是什么？
- 持久序列化(Persistent Serialization)是什么？基本原理是什么？
- 事件服务 EVT(Event Service)原理是什么？
- 消息服务 MSG(Message Service)如何提供？
- 日志服务 LOG(Log Service)底层设计原理是什么？
- 通知服务 NTF(Notification Service)如何工作？
- OpenSAF 架构中有哪些方面值得我们借鉴？是否能将其中比如持久序列化(Persistent Serialization)引入我们的 DServer 系统中？
- 其他框架与模型，如可用性管理框架 AMF(Availability Management Framework)、信息管理模型 IMM(Information Management Model)特点？

OpenSAF相关资料，可参考官方网站OpenSAF，SAForum。

(2) <Effective C++>培训

针对 team member 的培训，几个人轮流培训，该书 55 讲笔者大概讲 20 个 Item。认真做好培训材料，并整理为文章，如下。(该系列文章已整理出多篇，持续写作、整理中)

重读经典-《Effective C++》Item0：基本概念
重读经典-《Effective C++》Item1：视C++为一个语言联邦
重读经典-《Effective C++》Item2：尽量以const,enum,inline替换#define
重读经典-《Effective C++》Item3：尽可能使用const
重读经典-《Effective C++》Item4：确定对象被使用前已先被初始化

(3) 单元测试和 Mock 研究

关于该专题的学习，笔者主要关注以下问题。

- 软件测试基本概念？单元测试的基本概念？常用单元测试工具有哪些？
- XUnit 家族的测试工具，其基本原理是什么？
- TestSuite，TestCase，Test 之间有什么关系？如何组织这些 case？如何 run 这些测试？
- 如何快速地使用某种单元测试工具？如 CUnit, GoogleTest。
- Mock 的基本概念，什么是 Mock？Mock 什么？如何 Mock？
- Mock 的目的？使用笨重的方法如何 mock？如何通过编译(选项)进行 mock？这种方法有什么优缺点？
- 常用的 Mock 工具有哪些？适用于什么平台？其基本思想是什么？如 CMock，其基本原理？CMock 本身如何设计的？由什么写成的框架？(Ruby)
- 嵌入式软件领域常用的单元测试工具和 Mock 工具有什么？如 CUnit+Unity，CUnit+CMock。如何结合使用？一般步骤有哪些？

该专题的学习，写成文章如下，基本上都是使用方法的介绍，原理性的介绍较少。

google test简介

如何编译google test的例子?
如何使用google test写单元测试?
Linux平台如何编译使用Google test写的单元测试?
Win32 平台如何编译使用Google test编写的单元测试?
使用Google Test的一个简单例子
Cunit简介
Mock的基本概念和方法(续)
MOCK object-第 7 章
Mock的基本概念和方法

关于CMock的原理，可参考Functionality and Design of the CMock framework等，其他的参考资料，可参考笔者的文章Reference，或者google一下。

(4) GCC Coverage 研究

该专题主要针对 GCC-4.1.2 版本，研究其覆盖率测试工具(gcov/gcov-dump/lcov)的设计、原理、实现与应用，并写成如下文章，共计 11 篇文章。同时，也将 GCC-4.1.2 代码中的 gcov/gcov-dump 代码抽取出来，并进行一些修改，成为独立的 gcov/gcov-dump 程序。笔者在研究中考虑的问题可参考具体文章，此处不再赘述。相关资料可参考文章 Reference。

Linux平台代码覆盖率测试工具GCOV简介
Linux平台代码覆盖率测试工具GCOV的前端工具LCOV简介
Linux平台代码覆盖率测试工具GCOV相关文件分析
Linux平台代码覆盖率测试-GCC如何编译生成gcov/gcov-dump程序及其bug分析
Linux平台代码覆盖率测试-从GCC源码中抽取gcov/gcov-dump程序
Linux平台代码覆盖率测试-gcov-dump原理分析
Linux平台代码覆盖率测试-.gcda/.gcno文件及其格式分析
Linux平台代码覆盖率测试-GCC插桩前后汇编代码对比分析
Linux平台代码覆盖率测试-编译过程自动化及对链接的解释
Linux平台代码覆盖率测试-GCC插桩基本概念及原理分析
Linux平台代码覆盖率测试-基本块图、插桩位置及桩代码执行分析

合集请参考<深入浅出Linux平台代码覆盖率测试——原理、工具、分析(v1)>(66 页)。(提供下载)

gcov/gcov-dump 程序代码已经打包并上传到 CSDN 的资源频道，地址如下。
gcov-dump-1.0.tar.gz : http://download.csdn.net/source/3235106
gcov-1.0.tar.gz : http://download.csdn.net/source/3235119
gcov-tools-1.0.tar.gz: http://download.csdn.net/source/3235127

该研究内容计划写作 15 篇文章，还有 4 篇需要持续完成。

以上(3)、(4)两个研究任务主要针对公司未来 Hitachi TND(Telecommunication Network Division)项目 AMN6400 等系统实施单元测试所作的研究。

4. 一些文档

关于 SPON 系统，DePON 系统(包括 DServer 系统)的学习，笔者写了一系列的文档或文章，总结如下。

<SPON arthitecture>
<SPON study note>
<VLAN Experiments using SmartBits>
<IGMP Snooping Experiment>
<IGMP Proxy experiment>
<Link Aggregation experiment>
<IGMP log analysis during restoration from 3.4 on>
<Analysis of OLT3540 & OLCDV auto-reboot because of CFM stack overflow during Y1731 test>

<Multicast-SNMP interface analysis>
<why not error since the same name 'MD5Init'?>
<DFDB(Dynamic Forwarding DataBase) event report process>
<The process of ONU configuration file parsing & installing port to vlan>
<The process of requesting ONU dynamic DFDBs notification in cmmEventTftpCm>
<CMTS interfaces and CM interfaces in DServer System>
...

5. 工作心得

关于 2010 年工作心得，可以参考2010 心得。

“你要随时想着自己所做的事情是否能为公司带来价值，这是你能在这个公司长久发展的基础。”这是我老婆曾经给我说过的一句话，记录于此，也算作我的工作心得吧。同时也希望自己在未来的工作中时刻思考自己的工作方向和目标，做对公司有意义的事情。

凡事懂得感恩，要感恩给我提供平台的盛立亚公司，要感恩父母，感恩曾帮助过我的每个人；更要怀着感恩的心并用积极的态度面对工作，面对生活，面对未来。也应该感恩我的老婆，因为她的善良，更因为她那颗金子般的心。也要感谢上帝，让我们一起携手创造未来。

三. 研究与学习篇

1. 研究与学习回顾

(1) glibc 代码研究

主要是针对<程序员的自我修养-链接、装载与库>这本书所做的研究与学习，重点关注如下问题。

- 编译、链接过程
- 目标文件格式
- 可执行文件的装载过程
- 静态链接过程，符号解析与重定位

- 动态链接步骤与实现
- Linux 共享库的创建、安装、版本
- 堆与栈、堆的分配算法
- 函数调用栈分析
- 运行库如何实现 C++全局构造和析构

...

- POSIX 线程库

关于该专题，笔者已经完成的文章如下，其他文章有待继续整理。

<Is glibc or uclibc thread-safe>
<glibc 及其与 gcc 的关系简介>
<gcc, glibc, binutils 间的关系>

(2) STL 代码研究

研究对象是 SGISTL3.3，GCC-4.1.2 自带的 STL 这两个版本。尤其对其中的 List，heap，queue，deque，vector, set, map，rbtree 等数据结构设计和实现有较深入的研究，并通过例子跟踪调试，并动手实现大部分的数据结构。

笔者希望能整理出关于该专题的文章，让自己不忘这些经典数据结构的设计技巧和实现方法。

(3) boost::lexical, boost::multi_index_container 子库的研究

对 boost 的 overview 及 boost::multi_index_container 子库的原理、使用进行了比较深入的研究，可参考如下文章，共计 14 篇文章，且该专题也是“重构”的一个非常好的例子。

Boost简介
Boost库的命名规则
Win32 平台Boost的编译方法
Linux平台Boost的编译方法
Win32 平台如何配置才能使用Boost、ACE、CppUnit
Boost组件multi_index_container实例(1)
Boost组件multi_index_container实例(续 2)
Boost组件multi_index_container实例(续 3)
Boost组件multi_index_container实例(续 4)
Boost组件multi_index_container实例(续 5)
Boost组件multi_index_container实例(后续)
Boost组件multi_index_container组合查询实例(1)
Boost组件multi_index_container组合查询实例(续 2)
Boost组件multi_index_container组合查询实例(续 3)

该专题的研究主要是为了 DePON1.5 系统中 QoS 模块重构，其中对 MIB Table 数据结构的修改，先采用了 boost::multi_index_container，于是研究了一下该子库的使用方法；由于该设计不能很好的反映 table 间的关系，故后来改用链表(模板)来表示数据集本来的物理关系。

(4) Linux kernel 代码研究

主要关注以下几个方面，先从其基本原理，基本设计思想入手，然后逐个深入分析，并写成文章。这是笔者的目标。

- 内核数据结构
- 中断及中断处理
- 系统调用原理及其实现
- 进程管理、进程调度、进程地址空间
- IPC 通信、内核同步方法
- 虚拟文件系统
- 内存管理
- 页面缓存和页回写
- 内核调试

关于 Linux Kernel 方面的研究，笔者希望继续坚持研究，坚持原创。这将是笔者未来 3 年内学习重点。笔者坚信，对内核的深刻理解一定会让很多工作事半功倍。

(5) Nginx-1.0.4 代码分析

目前还处在 Nginx 的数据结构分析阶段，已经完成 pool，List，array，queue，hash 数据结构的分析，可参考如下文章。

nginx-1.0.4 源码分析—模块及其初始化
nginx-1.0.4 源码分析—内存池结构ngx_pool_t及内存管理
nginx-1.0.4 源码分析—数组结构ngx_array_t
nginx-1.0.4 源码分析–链表结构ngx_list_t
nginx-1.0.4 源码分析—队列结构ngx_queue_t
nginx源码分析–hash结构ngx_hash_t(v1.0.4)

笔者研究 Nginx 的目的是希望能该中学习其作者优秀的设计方法和实现技巧，同时也为了给自己竖立一个短期目标，充实自己的业余时间。

(6) 数学、智能、思维相关问题研究与学习

如网络上流传甚广的砝码分盐问题等思维问题的思考、解决和总结。关于该问题，笔者从数学和计算机的角度对其进行了全面的分析和解决，因太长，故分节如下。

砝码分盐问题—从数学和计算机的角度分析(1)
砝码分盐问题—从数学和计算机的角度分析(2)
砝码分盐问题—从数学和计算机的角度分析(3)
砝码分盐问题—从数学和计算机的角度分析(4)
砝码分盐问题—从数学和计算机的角度分析(5)
砝码分盐问题—从数学和计算机的角度分析(6)
砝码分盐问题—从数学和计算机的角度分析(7)
砝码分盐问题—从数学和计算机的角度分析(8)
砝码分盐问题—从数学和计算机的角度分析(9)

砝码分盐问题——从数学和计算机的角度分析(10)
砝码分盐问题——从数学和计算机的角度分析(11)

合集请参考<砝码分盐问题——从数学和计算机的角度分析>(提供下载)。

其他问题，如奇阶幻方，笔者从数学的角度分析其特性、解的通用性等，其他诸如中国余数定理、12 球问题、13 球问题、64 球问题等知识、智能开发相关的问题。另外，网络上有个经典的面试题目，即阅读论文A Sorting Algorithm for Polynomial Multiplication并实现其中算法，该论文讲述多项式乘积算法及其优化，笔者为其实现了前两个算法，并完成相关文档。

笔者对这类问题的研究，目的是为了让自己能够保持活跃的思维，以养成深入思考问题本质的习惯。

2．研究与学习结果

写这一节，我想我是可以自豪一下的。☺

工作和学习中使用 Onenote2007 做笔记，总计 1600 页(大约三分之一是工作笔记)。其中部分内容发到了 blog 上，希望能继续整理、发布。除此之外，笔者在平时的工作和学习中，养成了随身携带笔记的习惯，将自己的学习和思考、idea、工作计划、to-do-list、心得等随时记下，共计 13.3 本。

3．研究与学习心得

此处记录笔者研究开源代码的方法。

读源码，有一项重要的任务，就是搞懂数据结构，包括逻辑结构和物理结构，搞懂对象间的关系，搞懂数据的流向。关于这一点，看参考Nginx-1.0.4 源码分析的相关文章hash结构ngx_hash_t(v1.0.4)等。

此外，如下几个问题也应该考虑。

- 了解业务流程，设计，和软件架构
- 了解层次关系，如类层次、类关系、类依赖、类继承等
- 了解数据结构及数据流向，对象内存结构(最好的方法是 debug)
- 若有日志系统，打开日志功能(日志记录执行过程，分析方便)
- 还要会 compile，debug，install 等
- 学习其设计模式，设计方法，实现技巧

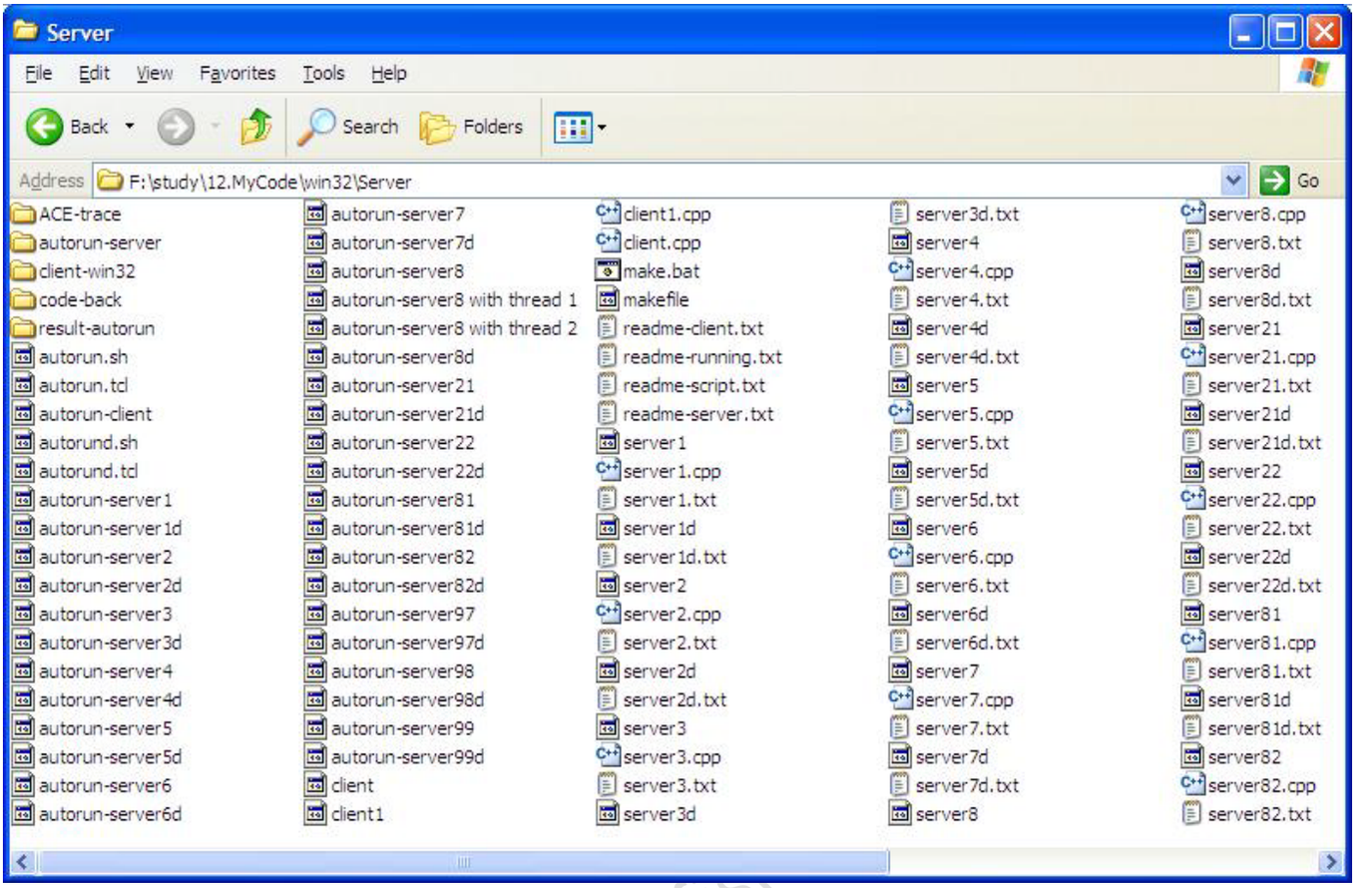
四．总结篇

1．对自己满意的

(1) 有较好的开发习惯，并坚持完善

建立自己研究与学习环境，对某一个问题、有新意的算法、疑点等，随时准备编写代码验证、观察、思考、总结。

如，关于 ACE_Reactor 的研究，编写代码，编写 makefile 自动编译所有程序，且使用 TCL 和 Expect 软件自动运行、测试所有程序，并自动保存所有程序的运行结果以便对比分析。如下图。



图中，如 server6d，表示 debug 版。

(2) 深入理解问题和所做工作，并坚持写文档

在此罗列一下写过的文章。

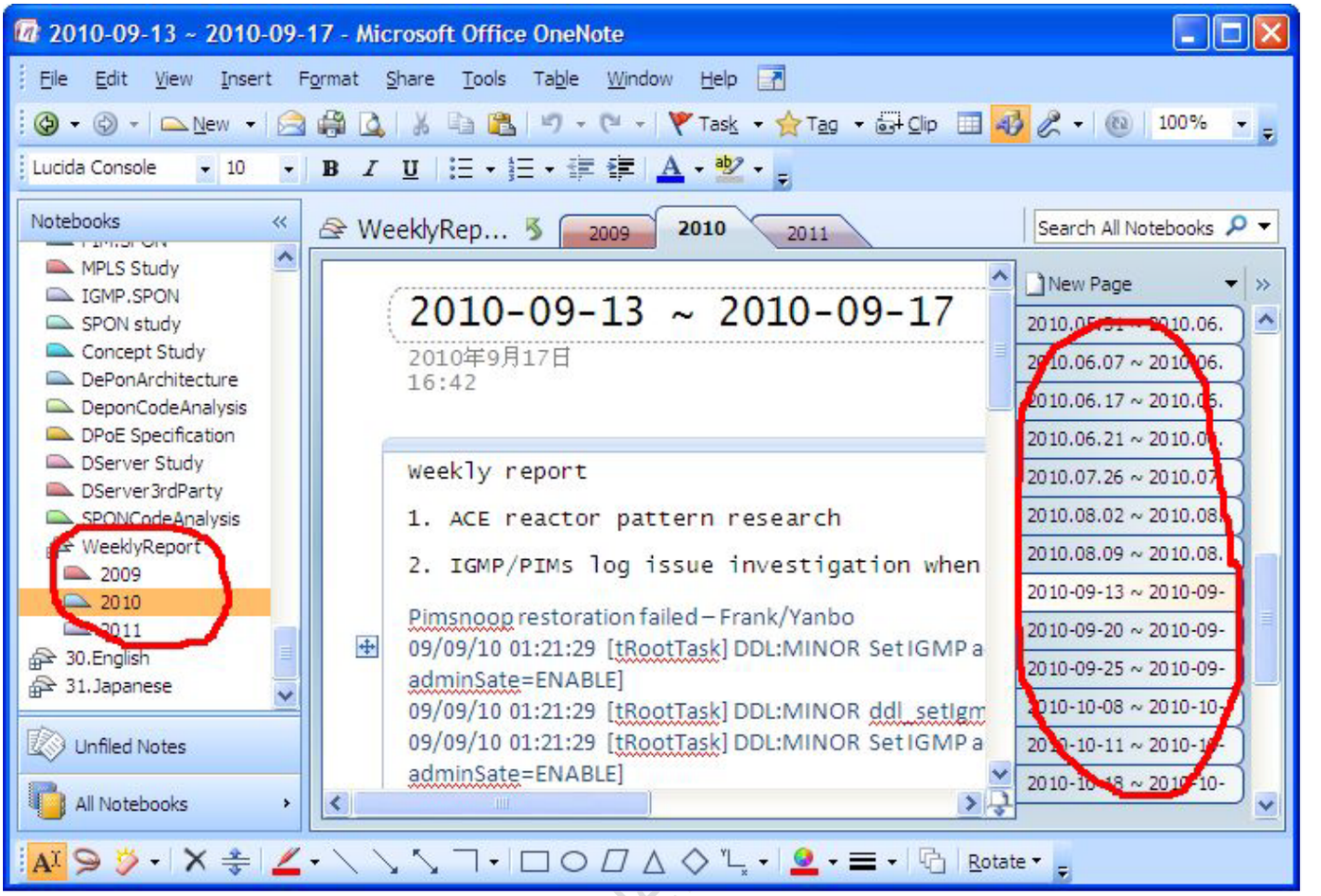
《深入理解计算机系统》3.38 题解—缓冲区溢出攻击实例 (22 页)
Explanation about “pure virtual function call” on Win32 platform (17 页)
Analysis of ACE_Task-putq with timeout=0 when queue is full on Linux platform (22 页)
使用GDB调试RB-tree的几个问题 (18 页)
使用GDB调试RB-tree的几个问题(更正) (6 页)

Mock的基本概念和方法 (20 页)
深入浅出Linux平台代码覆盖率测试—原理、工具、分析(v1) (66 页)
砝码分盐问题—从数学和计算机的角度分析 (28 页)
...

也可参考http://www.abo321.org/shareddoc(提供下载和在线阅读)。

(3) 坚持写 weekly report，坚持写工作文档

这是一个较好的工作习惯，可以让自己在一个月、半年、年终回顾工作时能一目了然地知道自己到底做了些什么。写过的工作文档，请参考前文。关于 weekly report，如下图。



(4) 坚持原创 blog，坚持分享

写作时都是按专题进行，一般一个专题写 10 篇左右的文章来探讨一个技术的方方面面。如上文所述的 boost::multi_index_container，ACE_Reactor，GCC Coverage 研究，Nginx-1.0.4 源码分析等。

2．对自己不太满意的

进步中也有不足，比如，研究和学习的东西有点杂，这样虽然自己的知识面比较宽，但直接结果就是不能特别专注，致技术不会太精、太深入。毕竟一个人的精力是有限的，一个人不可能成为全能的技术专家。这跟工作的要求有关，不同的任务可能有不同的技术涉及；跟自己的兴趣也有一定的关系。在未来的职业生涯中，应该继续坚持 Linux Kernel、软件架构等的研究，这是专注的焦点。

对自己的要求：专注，坚持。

3．感悟

以下这些话是我在 13.3 个笔记本上写下的，大部分在第一页或最后一页，中间偶尔得之，也会记录下来。记录于此，以备回顾、提醒自己。其中，部分是在网上读到的，深有感触，就记下了，或许笔者根据当时的感受做了修改，在此谢谢这些朋友。

你的生活中，什么是最重要的？

我的目标是什么？我现在在做什么？我做的与达到目标要做的相差多少？
在一家公司，你应该关心什么？才符合公司的发展，才符合自己的成长？
这个软件开发过程中，或者在这个项目中，我真的做了巨大贡献吗？

要认可自己的进步，并不断学习和完善自己。
技术、知识以及锻炼出来的经验，是你在与人对比中最重要的部分。
你今天的现状是你几年前选择的结果，你今天的选择决定你几年后的状况。要时刻提醒自己。
曾经的经历和学习都不重要，重要的是你的今天和明天，是不是在不停地提高自己，坚持终生学习，每天一小步即可。

人，归根结底是为自己工作，我们在工作中获得的经验、知识、能力、金钱、成长等，实际上为自己的将来在积蓄力量、积蓄资本。

负责什么，你就应该精通什么。
坚持做一件事。每个阶段的工作和学习，都坚持做一件事。
将自己的职业生涯当作一个产品来开发，考虑供给和需求，逐步建立个人品牌。

按人生目标、工作性质的相关要求学习和掌握知识，而非个人喜好。
艰苦卓绝的努力，从孜孜不倦地读书、实践、思考、总结、记录开始。
让记录成为一种习惯。在每个工作日，花一点时间考察、精炼自己的工作方法。
坚持积累，坚持总结。坚持研究自己，了解自己。深入挖掘一些工具、技术和技艺，让自己成为一个持续学习和思考的人。

踏踏实实做事，规规矩矩做人，快快乐乐生活。

Take it easy, because Nothing is impossible. Have a try, and Just do it, you will Get things done.
这就是我喜欢的 5 句简单的英文句子组成的话。

4．展望未来

800 天是短暂的，但我可以自豪的对自己说，在这 800 天的时间里，我并没有虚度，相反，我看到了自己的进步，看到了自己的成长——这或许是自信的基础吧。希望下次再写工作总结的时候，我可以写“这些年，我有了长足的进步!”。

祝福盛立亚，祝福岩波，华仔和盛立亚的其他同事。祝福未来我将服务的公司。祝福所有人。祝福父母，祝福老婆，也祝福自己。祝福儿子，继续健康成长，开心快乐。

仅用此文记住曾将给过我指导、关心和帮助的每一个人。希望自己每一天都带着感恩的心去做每一项工作、完成每一个任务，带着感恩的心去生活每一天。

关于我的工作经历，可以参考LinkedIn Profile。
有任何问题，可以发邮件(livelylittlefish@gmail.com)给我，谢谢。

余祖波
2011.7.29