Confessions of a Recovering Proprietary Programmer

（一个复原中的私有软件程序员的告白）

Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center / Linaro
Overview (概览)

- What Does Paul Know About Open Source? (关于开源软件保罗知道哪些)
- Parable of Six Penguins and the Elephant (六个企鹅和大象的寓言)
- Coding Style (代码风格)
- Source Code Management (源代码管理)
- Summary (结束语)
What Does Paul Know About Open Source?

（关于开源软件保罗知道哪些）
Who is Paul and How Did He Get This Way?

- Grew up in rural Oregon
- First use of computer in high school (72-76)
- BSME & BSCS, Oregon State University (76-81)
- Contract Programming and Consulting (81-85)

- IBM mainframe: punched cards and FORTRAN
  - IBM mainframe: 打孔卡和 FORTRAN
- Later ASR33 TTY and BASIC
  - 后是 ASR33 TTY 和 BASIC
- Tuition provided by FORTRAN and COBOL
  - 学费来源于 FORTRAN 和 COBOL
- Building control system (Pascal/z80)
  - 控制系统
- Security card-access system (Pascal/RT11/PDP-11)
  - 门禁系统
- Dining hall system (Pascal/RT11/PDP-11)
  - 食堂系统
- Acoustic navigation system (C/BSD2.8/PDP-11)
  - 声学导航系统
Who is Paul and How Did He Get This Way?

28 周年：1983 年五月至今
Who is Paul and How Did He Get This Way? ( 谁是保罗，他是如何做到的 )

- **SRI International (85-90) (SRI 国际公司)**
  - UNIX systems administration (BSD/Pyramid90x) (系统管理)
  - Packet-radio research (C/SunOS/68K) (分组无线电研究)
  - Internet protocol research (C/SunOS/SPARC) (互联网协议研究)

- **Sequent Computer Systems (90-00) (Sequent 公司)**
  - Communications performance (C/DYNIX-ptx/x86) (通信性能)
  - Memory allocators, TLB, RCU, timers, … (内存分配, TLB, RCU, 定时器)

- **IBM (00-present)**
  - NUMA-aware and brlock-like locking primitive in AIX (AIX 中 NUMA 知晓的和类似 brlock 的加锁原语)
  - RCU maintainer for Linux kernel (内核 RCU 的维护者)
What Does Paul Know About Open Source? (关于开源软件保罗知道哪些)

- Early member of IBM's Linux Technology Center (IBM Linux 技术中心的早期成员)
  - Helped define IBM's open-source strategy (帮助制定 IBM 的开源战略)
- Active contributor to the Linux kernel: (活跃的内核贡献者)
  - 379 patches accepted into mainline since 2005 (05年到今共379个)
    - 1878 from gregkh, 2185 from tglx, 2592 from viro, 3395 from mingo, 3841 from davem, 10,411 from torvalds
  - Maintainer of read-copy update (RCU) (RCU 的维护者)
- Recognized expert in Linux community for concurrency, memory ordering, and RCU (社区里在并行、内存序、RCU公认的专家)
  - One of a very few people to invent a synchronization primitive (RCU) with order-of-magnitude performance benefits that has been accepted into the Linux kernel (发明一种同步原语，带来 10 倍以上的性能提升，并被内核所接受)
  - Numerous concurrency experts won't be forgiving him for this any time soon... :-)

"Confessions of a Recovering Proprietary Programmer" for Linaro Connect
How Much is RCU Used in the Linux Kernel?
(RCU 在内核中使用情况)
Paul is a Recovering Proprietary Programmer
(Paul 是一个复原中的专有软件程序员)

The Parable of
The Six Blind Penguins and the Elephant
(瞎子摸象的故事)
Proprietary Programming: Requirements
（专有软件的编程：需求分析）
Proprietary Programming: “Solution”
（专有软件的编程：解决方案）
Example: DYNIX/ptx RCU Implementation
(例子：DYNIX/ptx 中的 RCU 实现)

- In late 1990s, I knew everything there was to know about RCU: (90年代后期，我认为我知道关于 RCU 的所有东西)
  - RCU read-side critical sections (RCU 读者临界区)
    - “Free is a very good price!!!”
  - RCU grace periods (RCU 宽免期)
  - RCU quiescent states: context switches, CPU idle, syscall entry, trap entry, CPU offline (RCU 宁静态)
  - rcu_read_lock(), rcu_read_unlock(), read_barrier_depends, call_rcu(), kfree_rcu() (RCU 的调用接口)
  - RCU application to lists, hash tables, trees, mode change, and waiting for ongoing interrupts (RCU 的应用：列表 哈希表，树 ...)
  - Impressive performance and scalability benefits for UNIX-based database servers (基于 UNIX 的数据库上显著的性能和可伸缩性优势)
    - 64 CPUs SMP, 256 CPUs clustered
Proprietary Programming: “Solution”
(专有软件的编程：解决方案)

But sooner or later...
The Entire Elephant Will Make Itself Known...
(大象会让别人知道它的整体到底是什么)
What I Didn't Know About RCU in the 1990s:
(我在90年代所不知道的RCU的其他扩展)

- DoS attacks
- Energy conservation
- Real-time response
- Sleeping RCU readers
- Wait for callbacks: rcu_barrier()
- RCU list primitives
- Burying memory barriers into RCU primitives
- Handling DEC Alpha
- Handling value speculation
- RCU semantics
- RCU proofs of correctness
- Runtime RCU validation
- Static RCU validation
- Handling more than 64 CPUs
- RCU priority boosting
- Early-boot RCU uses
- RCU tracing
- RCU and type-safe memory
- User-level RCU
- Multi-tail callback lists
- rcutorture
- Single-CPU implementations
- RCU-protected atomic list move
- Resizable RCU hash tables with wait-free readers
- Workqueue-based RCU
- Expedited grace periods

What does the red font signify?
What does the yellow font signify?
So What Happened? (发生了什么事)

- Yes, I was and am the world's expert on RCU
- But I learned a lot about RCU from newbies in the Linux community
- It was well worth wading through their naïve and silly suggestions to get the benefit of some extremely valuable ideas
  - Which are listed in red on the previous slide
    - Many of which I would never have thought of
  - The yellow items are things that I implemented, but in response to situations brought to my attention by RCU newbies
    - Situations that I never would have imagined myself: “why would you need that?”
FOSS Programming: Requirements
Just Another Day on LKML...
But Sometimes Consensus is Achieved
And an Appropriate Solution Produced Thereby
This is RCU in DYNIX/ptx

- `rcu_read_lock()
- `rcu_read_unlock()`
- `rcu_assign_pointer()` [Sort of]
- `kfree_rcu()
- `call_rcu()`
This is RCU in DYNIX/ptx

rcu_read_lock()
rcu_read_unlock()

rcu_assign_pointer() (Sort of)

kfree_rcu()
call_rcu()
This is RCU in Linux

For legible version, see: http://lwn.net/Articles/418853/
Without Contributions From Linux Community:

- **Use of RCU would be error-prone:**
  - Burying memory barriers into RCU primitives
  - Runtime RCU validation
  - Static RCU validation
  - RCU semantics
  - RCU proofs of correctness

- **RCU would not be robust:**
  - DoS attacks

- **RCU would fail to handle important use cases:**
  - Sleeping RCU readers
  - Early-boot RCU uses
  - Wait for callbacks: rcu_barrier()
  - RCU and type-safe memory
  - User-level RCU
  - Resizable RCU hash tables with wait-free readers
  - Workqueue-based RCU

- **RCU would be slow and energy-inefficient**
  - Expedited grace periods
  - Multi-tail callback lists
  - Energy conservation (dyntick-idle mode)
Lessons Learned From the RCU Experience
(从RCU经历我们学到了什么)

- Linux runs an incredible variety of workloads
  (相当多种类的平台和软件运行在Linux)
  - Embedded, realtime, desktop, network, server, supercomputer...
  - Your solution might be perfect for embedded, but bad elsewhere

- Linux powers significant networking infrastructure
  (Linux提供了重要的网络基础设施)
  - You can't hide behind a firewall: Linux is the firewall

- Linux runs realtime workloads: Realtime effects are pervasive
  (Linux运行实时应用)

- Very large number of kernel developers (thousands)
  (千千万万的内核开发者)
  - If one person year of work saves 1% of everyone's time:
  - Linux: ~10,000 developers gives ~100 person-years per year payback
    - Investment pays off in less then four days
    - Even if only 500 full-time-developer equivalents, payoff in about 10 weeks
  - Proprietary: ~40 developers gives ~0.4 person-years per year payback
    - Investment takes more than two years to pays off

- Code developed in specialized environments will need serious modifications!!!(在专有环境下开发的代码需要相当的修改)
Lessons Learned From the RCU Experience
(从RCU经历我们学到了什么)

- The Linux kernel community probably does not know who you are or what you are capable of (社区不知道你是谁和你的能力)
  - You will need to prove yourself to them (你需要在社区里证明你自己)
  - Just as you would to any new community you were to join
  - Time spent learning about the community is time well spent (花在学习社区本身上的时间是值得的)
    - LWN articles, mailing-list archives, ...
- Respond quickly: hours or days, not weeks or months (尽快回复：以小时和天计，不要数周或者数月)
- Maintain a professional bearing and attitude (保持职业的风度和态度)
  - If flamed, respond to the technical points, not to the emotion
    - The irritation is momentary, but an ill-considered reply is archived forever
  - It sometimes takes some effort to tease out technical points
Other Examples of Good Solutions

▪ Dynticks
  ▪ Better consolidation on mainframes
  ▪ Better battery life on embedded devices
▪ Real-time Linux
  ▪ Changes that improve real-time response...
  ▪ ... often improve scalability on multicore systems
▪ Group scheduling
  ▪ Helps servers manage their workloads
  ▪ And also helps kernel hackers get good response times during large kernel builds
But Sometimes Things Get Stuck Here
But Sometimes Things Get Stuck Here: Android!
Confessions of a Recovering Proprietary Programmer

Maintainership Structure,

or

“Why Do Those Idiots Keep Rejecting My Stuff?”
Linux Kernel Structure: People

Linus Torvalds

Maintainers

Sub-maintainers

Developers

Source Tree

Code
Linux Kernel Structure: People

- Linus Torvalds
  - Source Tree
  - Code
  - Testing, Validation, and Q/A Here

- Maintainers
- Sub-maintainers
- Developers
If I accept an RCU patch, then I am taking responsibility for it.
Linux Kernel Structure: People

If I accept an RCU patch, then I am taking responsibility for it. And the same thing applies to my upstream maintainer.
If I accept an RCU patch, then I am taking responsibility for it. And the same thing applies to my upstream maintainer. So we reject patches with quality/maintainability problems.
If we did otherwise, the Linux kernel would be a complete mess.
Confessions of a Recovering Proprietary Programmer

Coding Style, or “When in Rome...”
Confessions: Coding Style
（告白：代码风格）

- “do {} while (0)” for statement-like cpp macros
  - But static inline functions instead whenever possible
  - Exceptions: polymorphic, iterators, declarators
- 80-column line limitation
- 8-space tabs
- No trailing space on lines
- Memory barriers must always be commented
- “return foo;”, not “return (foo);”
- Omit unnecessary parentheses
- No braces for one-line “then” or “else” clauses
- No “#if” or “#ifdef” in .c files
Confessions: Coding Style: #ifdef
(告白：代码风格之 #ifdef)

- I had been using #ifdef wherever I felt like it for two decades
  - Why change?
- Started a new project
  - Smallish, so simply wrote it both ways
  - Quickly abandoned the #ifdef-in-.c version
  - Why?

- Sometimes the Romans are right!
Confessions: Coding Style
（告白：代码风格）

- “return foo;”, not “return (foo);”
  - Fewer characters, more likely to fit in 80-characters
- Omit unnecessary parentheses
  - Fewer characters, more likely to fit in 80-characters
  - Good exercise for one's memory, I guess...
- No braces for one-line “then” or “else” clauses
  - Fewer lines, more likely to fit on single screen
  - But it does get me in trouble reasonably often

- Key point: Linux kernel is read far more often than it is written and/or debugged
  - The needs of the many readers outweigh those of the few(er) writers and debuggers
Confessions of a Recovering Proprietary Programmer

Source-Code Management
Confessions: Source-Code Management

（告白：代码管理）
Confessions: Source-Code Management
（告白：代码管理）
Confessions: Why so Primitive???

(告白：为什么这么原始)
Confessions: Source-Code Management
（告白：代码管理）

- Revision Control System (RCS)
  - Created by Walter Tichy in 1980s
  - I used it for about 20 years
- Strong Points:
  - Less need to retype old versions from printouts
  - Trivial to track changes on file-by-file basis
- Shortcomings:
  - Hard to get consistent past view of set of files
  - Hard to use collaboratively
    - Lots of different script wrappers for RCS to make this work
  - Merges are extremely painful and easy to get wrong
Confessions: Source-Code Management (告白：代码管理)

- **bitkeeper**
  - Created by Larry McVoy in late 1990s
  - I used it very lightly for a few years

- **Strong points:**
  - Consistent global view of source tree
  - Better support for merges
  - Easier to use collaboratively

- **Shortcomings:**
  - Proprietary software
  - Massive political hassles
Confessions: Source-Code Management (告白：代码管理)

- **git**
  - Created by Linux community in mid-00s

- **Strong points:**
  - Consistent global view of source tree
  - Way better support for merges: difference in kind
  - Easier to use collaboratively
  - Integrates patch handling and maintainer roles
  - Automated branch rebasing

- **Shortcomings:**
  - Learning curve!!!
  - Don't try this on 1990s storage hardware...

- From “I hate git” to reasonably happy git user
Summary
(结束语)

- Paul knows something about open source
  - But don't take my word for it, ask Google!!!
  - And a lot of other Linaro folks know even more
- Open discussion often produces better results than isolated development
- Open-source development has surprising implications on coding style
- Open-source software has resulted in great advances in source-code management
Summary: Additional Material
(附加材料)

- Greg Kroah-Hartman's “Write and Submit your first Linux kernel Patch” (2010)
- Jonathan Corbet's “How to Participate in the Linux Community” (2008)
  - http://ldn.linuxfoundation.org/how-participate-linux-community
- Randy Dunlap's “Linux Kernel Development: Getting Started” (2005)
  - http://lwn.net/Articles/160191/
- Linux kernel documentation
  - Documentation/SubmittingPatches
  - Documentation/SubmitChecklist
  - Documentation/SubmittingPatches
Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
- This material is based upon work supported by the National Science Foundation under Grant No. CNS-0719851.
Questions?
Backup