Unix 文化与哲学

曹东刚

caodg@sei.pku.edu.cn

Linux 程序设计环境

http://c.pku.edu.cn/

内容提要

工程领域的文化

- 大多数工程领域都有自己的技术文化
  - 不成文的知识，传统，约定，...
  - 口口相传
- 软件技术领域有所不同
  - 技术变化太快，缺少稳定的技术文化
- Unix 是软件技术领域中的特例
  - 设计哲学，编程艺术，技术文化

Unix 的生命力

- Unix 具有极强的生命力和适应力
  - 历史最长，支持的硬件平台最多，应用领域最广泛，...
  - Unix 的核心知识经年不变：语言，系统调用，工具
- 为什么 Unix 具有如此的生命力？
  - Unix 诞生之初就具有的优良设计

对 Unix 的异议

- Unix 的文件模型
- Unix 的安全机制
- Unix 面向技术用户，而不是终端用户
- Unix 的“策略与机制分离”思想
  - 可选择的工具太多，工具的选项太多
  - 策略并不稳定，机制是长久的

Unix 带给我们

- 丰富的开源软件
- 活跃的开源社区
- 平台可移植性和开放标准
- Internet 和 WWW
- 高度灵活性
- 趣味性和可玩性
- 宝贵的经验和教训

内容提要

基本

- 源起 Ken Thompson 设计一个服务接口简洁、小巧精干的操作系统
- 非正规的科班开发方法
- 自底向上, 注重实效, 倚重经验

Doug McIlroy 的评论

- 让一个程序只做好一件事
- 程序要能协作
- 程序要能处理文本流

Rob Pike 从 C 编程角度的阐述

- **1:** 程序的性能瓶颈不能靠猜测得出来
- **2:** 只有度量之后才可开始优化
- **3:** 花哨的算法在 n 很小时通常很慢，而 n 通常是很小的
- **4:** 花哨的算法比简单的算法 Bug 更多，更难维护
- **5:** 数据结构才是编程核心
- **6:** 结束.

Unix 哲学概览

No.	Rule of	No.	Rule of
1	Modularity	10	Least Surprise
2	Clarity	11	Silence
3	Composition	12	Repair
4	Separation	13	Economy
5	Simplicity	14	Generation
6	Parsimony	15	Optimization
7	Transparency	16	Diversity
8	Robustness	17	Extensibility
9	Representation		

模块化原则

- “计算机程序设计的本质是控制复杂性” — Brian Kernighan
- 软件维护至关重要且代价高昂
- 代码模块性的发展
  - 大块机器代码 $\Longrightarrow$ 子例程 $\Longrightarrow$ 库 $\Longrightarrow$ 进程 $\Longrightarrow$ 分布进程
  - 然而，“没有银弹” — Fred Brooks
- 通过模块化降低整体复杂度

复杂度

- 复杂度的来源
  - 实现复杂度
  - 接口复杂度
  - 代码量
- 接口复杂度和实现复杂度的折中
- 本质的、可选的、偶然的复杂度

复杂度的来源与映射

偶然复杂度	违反SPOT原则	过早优化	非正交性
可选复杂度	方法学开销	别的一切	有效功能
本质复杂度	开发工具	核心数据结构	功能需求
	代码库规模	实现复杂度	接口复杂度

清晰原则

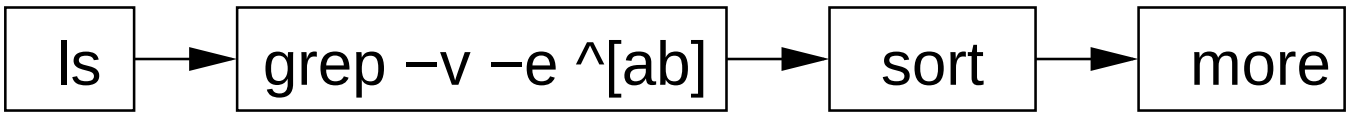
清晰胜于机巧

- 为程序员的理解、而不是机器的执行 —编写代码
- 良好的代码注释
- 算法尽量简单

组合原则

设计时考虑组合可组合的程序或构件

- 程序间的通信协议尽量为简单的文本流格式



Unix pipe-filter pattern

- 程序互相独立
- 谨慎对待 GUI, 尽量避免人机过多交互

分离原则

策略与机制分离; 接口与引擎分离

- 提供引擎, 将策略留给应用

  如: X window, GNU Hurd/Mach micro-kernel

- 用脚本包装服务函数库

  如: Emacs Lisp and C library primitives

- 前后端进程通过 socket 通信

  如: PostgreSQL server and client

简洁原则与吝啬原则

- 抵制编写错综复杂程序的诱惑
- 鼓励 简洁为美的软件设计文化
- 除非必要, 绝不编写庞大的程序

透明性原则

设计要可见，以便审查和调试

- 透明性：用户可以一眼看出软件在做什么以及如何做
- 可见性：软件系统包含了帮助人们理解软件的功能
  - 文档，注释，好的变量名和函数名
  - 在设计之初就考虑加入调试功能

透明性和可见性使得程序便于审查和调试

如何满足透明性

优雅的设计

- 使用简单的输入输出格式
- 设计简单的接口
- 避免过分的抽象, 如过深的对象继承
- 容易找到给定函数的代码
- 静态调用深度最好不超过 4
- API 的函数最好正交

健壮性原则：源自透明与简洁原则

软件不仅要能在正常条件下运行良好，在异常情况下也应能应对.

- 保持程序的内部逻辑易于理解
- 承受大量、极端的输入

表示原则

- 数据比算法容易驾驭
- 如果无可选择

  倾向: 复杂的数据结构和简单的算法

  避免: 简单的数据结构和复杂的算法

通俗原则

接口设计应避免标新立异

- 切合用户已有的知识和习惯
- 倾听和该程序相关的人员的意见
- 尊重传统和惯例
  - Unix 配置文件格式约定，命令行开关选项，等

缄默原则

沉默是金: 如果程序没什么好说的, 就应保持沉默

- 历史: 资源宝贵
- 技术: 有利组装

补救原则

出现异常时应马上退出并给出足够的错误信息

- 尽可能优雅地处理输入错误和自身运行错误
  - 若不能，则尽量给出充分的诊断信息后及早退出
- 接收数据宽容，发送数据严格

经济原则

宁花机器一分，不花程序员一秒

- 不要重复发明轮子，学会复用
- 用高级语言编程
- 让机器做底层的编程工作

生成原则

用程序生成程序

- 例如: Parser/lexer 生成器, GUI 生成器, makefile 生成器,...

优化原则

跑之前先学会走

- *Kernighan & Plauger*: 90% 的功能现在能实现，比 100% 的功能永远实现不了强
- *Donald Knuth*: 过早优化是万恶之源
- *Kent Beck*: 先求运行，再求正确，最后求快
- *Ken Thompson*: 我最有成效的一天就是扔掉了 1000 行代码

多样原则

- 即使是最出色的软件也会受限于设计者的想象力
  - 没有所谓的不二法门或永远适用的解决方案
  - 软件不应该是封闭、僵化的
- Unix 传统奉行：多语言，开放的扩展系统，用户定制机制

扩展原则

- 为数据格式和代码留下扩展空间
- 设计的协议或文件格式应具备自描述性
- 应用程序的结构应组织良好，便于扩展功能

应用 Unix 哲学

- 来源和目标独立的过滤器
- 文本数据流
- 前端和后端清晰分离
- 用解释型语言构建原型，之后才用 C 编程
- 宽收严发
- 小即是美
- ...

内容提要

模块性

保持清晰，保持简洁

- Unix 一直倡导模块化，即使在其早期
  - Dennis Ritchie 的小把戏
- 评价模块的代码质量
  - 封装性，紧凑性，正交性

编写复杂软件的唯一方法是编写简单的各个部分，然后通过清晰的接口将之连接起来

封装性

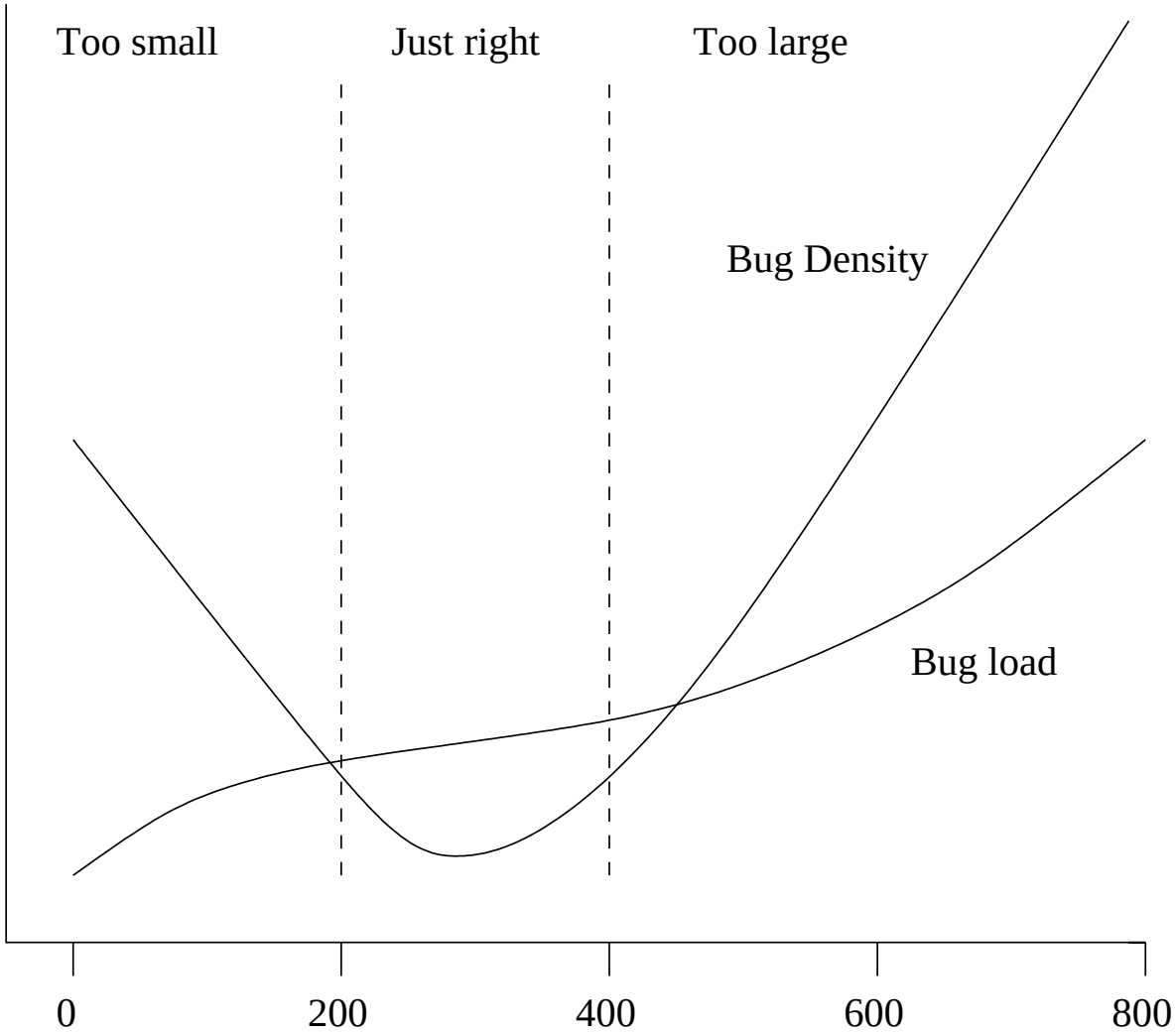
良好定义的模块在封装性方面

- 不会对外暴露内部细节
- 不会调用其他模块的内部实现
- 不会胡乱共享全局数据
- 通过良好设计的 API 进行通信

如何验证 API 是否设计良好

- 能否用自然语言描述清楚?
- 养成首先为接口编写注释的习惯

最优的模块大小


Too small
Just right
Too large
Bug Density
Bug load
0
200
400
600
800

紧凑性

紧凑性：设计能否装入人脑的特性

- 如何评价设计的紧凑性：有经验的用户是否需要手册
- 如何评价 API 的紧凑性：7 ± 2 法则

程序设计工具的紧凑性

- 紧凑: make, man
- 半紧凑: c, python, Unix 系统调用
- 非紧凑: perl, java, shell
- 反紧凑: c++

正交性

正交性有助于使复杂设计变得简单

- 正交系统的操作无副作用
- 如果不增加复杂性、引入 Bug，特定的副作用也可以接受
- 正交设计的例子：Unix API
- 非正交设计的例子：BSD sockets，X Window
- 避免依赖其它函数的副作用

SPOT 法则

Single Point Of Truth, or "Don't Repeat Yourself"

- 任何知识点在系统内都应该有一个唯一、明确、权威的表述.
- 重复可能会导致矛盾、产生隐藏问题.
- 常量、表、元数据只应声明和初始化一次.
- 使用代码生成工具消除重复

围绕问题明确的强力算法进行设计

良好形式化的例子

diff	顺序比较
grep	正则表达式
lex	非确定有限自动机
yacc	LR(1) 文法

启发性算法的例子

- 垃圾邮件过滤
- 虚拟内存管理

BSD Socket 示例

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <unistd.h>
9
10 int main(void)
11 {
12     struct sockaddr_in stSockAddr;
13     int SocketFD = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
14     if(-1 == SocketFD) {
15       perror("can not create socket");
16       exit(EXIT_FAILURE);
17     }
18     memset(&stSockAddr, 0, sizeof(struct sockaddr_in));
19     stSockAddr.sin_family = AF_INET;
20     stSockAddr.sin_port = htons(1100);
21     stSockAddr.sin_addr.s_addr = INADDR_ANY;
```

BSD Socket 示例 (cont)

```
22      if(-1 == bind(SocketFD,(const struct sockaddr *)&stSockAddr, sizeof(struct
            sockaddr_in))) {
23        perror("error bind failed");
24        close(SocketFD);
25        exit(EXIT_FAILURE);
26      }
27      if(-1 == listen(SocketFD, 10)) {
28        perror("error listen failed");
29        close(SocketFD);
30        exit(EXIT_FAILURE);
31      }
32      for(;;) {
33        int ConnectFD = accept(SocketFD, NULL, NULL);
34        if(0 > ConnectFD) {
35          perror("error accept failed");
36          close(SocketFD);
37          exit(EXIT_FAILURE);
38        }
39        shutdown(ConnectFD, SHUT_RDWR);
40        close(ConnectFD);
41      }
42      close(SocketFD);
```

讨论

Discussion

比较 Unix 和 Windows 文件操作 API，谁的正交性更好？为什么？

如何帮助提高模块性

- 全局变量的数目
- 模块大小
- 单个函数的规模
- 有无内部 API
- API 的参数
- 模块的入口点如何分布

内容提要

The Unix Philosophy in One Lesson