应用视角的操作系统

指令序列和高级语言的状态机模型；

目录

什么是软件（程序）？

操作系统眼中，程序就是状态机。

汇编代码的状态机模型

Everything is a state machine: 计算机 = 数字电路 = 状态机

- 状态 = 内存 M + 寄存器 R
- 初始状态 = ABI 规定（例如有一个合法的 %rsp）
- 状态迁移= 执行一条指令
  - gdb 同样可以观察状态和执行

    > starti: 从第一条指令开始

操作系统上的程序

- 所有的指令都只能计算
  - deterministic: mov, add, sub, call, ...
  - non-deterministic: rdrand, ...
  - syscall 把 (M,R) 完全交给操作系统

因此，我们对操作系统上程序的一个很重要的理解是程序是计算和系统调用组成的状态机；大部分计算指令都是确定性（deterministic，在相同环境下执行的结果总是相同）的，少部分指令（如 rdrand 返回随机数）则有非确定的结果。系统调用指令是非确定性的，操作系统可能会将计算机运行环境中的信息，例如来自设备的输入传递到程序中。

简单 C 程序的状态机模型（语义）

状态

- Stack frame 的列表 + 全局变量

初始状态

- 仅有一个 frame: `main(argc, argv)` ；全局变量为初始值

状态迁移

- 执行 frames.top.PC 处的简单语句
- 函数调用 = push frame (frame.PC = 入口)
- 函数返回 = pop frame

理解编译器

我们有两种状态机

- 高级语言代码：`.c`
  - 状态：栈、全局变量；状态迁移：语句执行
- 汇编指令序列：`.s`
  - 状态：(M,R)；状态迁移：指令执行
- 编译器是二者之间的桥梁：
  - `.s = compile(.c)`

操作系统中“任何程序”的一生

> 任何程序 = minimal.S = 调用 syscall 的状态机

- 被操作系统加载
  - 通过另一个进程执行 `execve` 设置为初始状态
- 状态机执行
  - 进程管理：`fork, execve, exit, ...`
  - 文件/设备管理：`open, close, read, write, ...`
  - 存储管理：`mmap, brk, ...`
- 调用 `_exit` (`exit_group`) 退出

总结

Take-away Messages

无论是汇编代码还是高级语言程序，它们都可以表示成状态机：

- 高级语言代码 `.c`
  - 状态：栈、全局变量；状态迁移：语句执行
- 汇编指令序列 `.s`
  - 状态：(M,R)；状态迁移：指令执行
- 编译器实现了两种状态机之间的翻译

应用程序与操作系统沟通的唯一桥梁是系统调用指令（例如 x86-64 的 syscall）。计算机系统不存在玄学；一切都建立在确定的机制上

- 理解操作系统的重要工具：`gcc, binutils, gdb, strace`

课后习题/编程作业

1. 阅读材料

- 浏览 GNU Coreutils 和 GNU Binutils 的网站，建立“手边有哪些可用的命令行工具”的一些印象。
- 浏览 gdb 文档的目录，找到你感兴趣的章节了解，例如—“Reverse Execution”、“TUI: GDB Text User Interface”……
- 对于你有兴趣的命令行工具，可以参考 busybox 和 toybox 项目中与之对应的简化实现。Toybox 现在已经成为了 Android 自带的命令行工具集。

> 查看man strace 一个有用的系统工具

2. 编程实践

在你的 Linux 中运行课堂上的代码示例，包括：

- 编译链接最小二进制文件，并使用 strace 查看执行的系统调用序列。

3. 关于vim的小Tips

```
strace -f gcc a.c &| vim -

vimrc(command):
    :set nowrap
    :%!grep -e execve -e open
```