实现操作系统的机制

蒋炎岩

南京大学 | 计算机软件研究所 | 系统与软件分析研究组

实现操作系统的机制：C程序视角

机制(mechanism)与策略(policy)分离

- 机制：提供做某些事的手段
  - 进程抽象(由中断驱动的寄存器现场切换)
  - 虚存抽象(维护$VM(x)$映射)
  - x86 (提供访问硬件的指令集)
  - 其他大家熟悉的机制：中断机制、消息机制……

- 策略：事情的具体做法
  - 在什么时候切换？切换到哪个进程？
  - 如何分配/映射物理页？
  - 怎样选择进程在哪个处理器上执行？

例子

- 为什么要分离机制和策略?
  - 机制是一个比较小但灵活可扩展的“内核”
  - 基于机制可以实现各种策略，满足应用需求

游戏提供了可玩的机制，你在此基础之上，用自己的策略获得高分。
女生提供了可追的机制，你在此基础之上，用自己的策略俘获芳心。
Unix提供了可用的机制，你在此基础之上，用自己的策略完成程序。

——知乎大神

操作系统是一个C程序

- C程序是从main开始执行的
  - 对于我们编写的程序来说，是由操作系统加载的
  - 第一条指令位于操作系统映射的ld-linux.so
  - 最终调用_start，libc初始化，才调用main

- 如果没有操作系统，谁来完成这一切？
  - 操作系统首先需要加载执行的机制

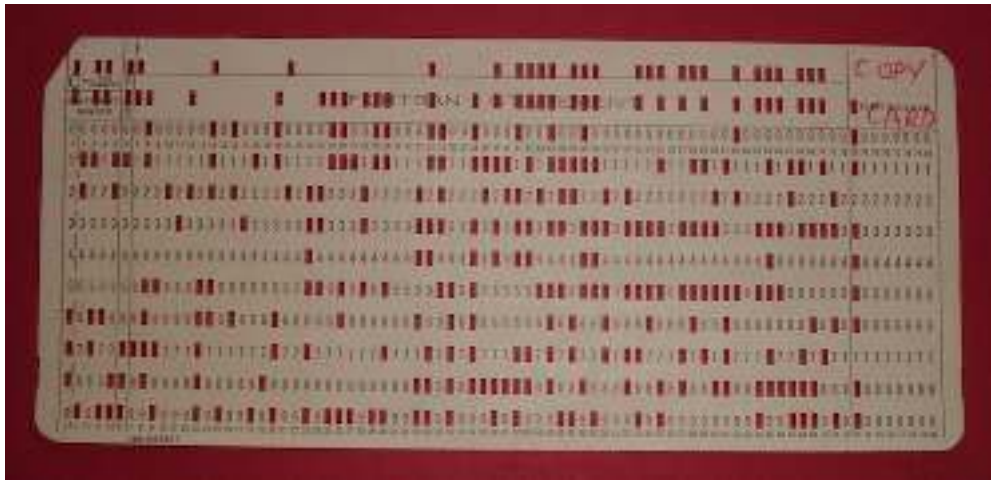
没有操作系统时的C程序

- 动态链接什么是(暂时)别想了

- 操作系统实验中的编译选项
  - -static
  - -O2 // 当然要优化啦
  - -m32 -march=i386
  - -fno-builtin // 不要builtin
  - -fno-pic // 不要PIC
  - -fno-stack-protector // 不要stack protector (?)
  - -fno-omit-frame-pointer // 不要忽略frame pointer (?)
  - (C++) -ffreestanding -fno-rtti -fno-exceptions

让“操作系统”这个C程序运行

- 硬件帮我们一点忙就好啦！
  - 只要硬件能帮我们执行哪怕一小段(我们写好的)程序
  - 我们就能用它做更多的事了(比如把操作系统载入内存)



早期的计算机系统启动：
计算机上有一个“Load”按钮
加载第一个打孔卡片上的程序执行

MIPS32

- 系统Reset后会从0xbfc00000开始执行指令

- MIPS32使用内存映射的I/O
  - 将这部分映射到ROM
  - 就获得了系统的控制权

PC启动 (1)

- CPU Reset后，处于16位模式，IP = 0xffff0
  - 你可以在i386手册上看到这一行为

- 这部分内存被映射到固件的代码(通常是一个跳转)
  - 然后固件执行一大堆一大堆一大堆一大堆一大堆一大堆事情
  - 多到简直是个操作系统

PC启动 (2)

- 在一大堆事情里，很重要的是寻找能启动的设备
  - 按顺序(BIOS配置)读出设备的第一个扇区(512字节)
  - 如果最后两个字节是0x55 0xAA，说明这是一个Master Boot Record (MBR)
    - *(uint16_t *)(buf+510) == 0xaa55 为什么?
  - 这是一个可以启动的设备
    - 把buf加载到内存0x7c00 – 0x7e00
    - 设置CS:IP = 0x7c00，主引导程序开始执行
    - 通常会加载一个更复杂的引导程序

UEFI

- 系统启动要做的事情很多
  - 与各种设备交互(磁盘、键盘、显示控制器……)
  - (否则你怎么选择启动什么设备)?
  - 兼顾安全性(固件又要能改又不能乱改，例如有名的CIH病毒)

- 这俨然就是个操作系统
  - 所以有了UEFI (Unified Extensible Firmware Interface)
  - 使用GUID管理磁盘(不再是MBR中的分区表)

历经千辛万苦

- 终于加载了512字节的代码能运行，这512字节代码做什么呢？
- 操作系统实验：最简单的情况

- 所以bootloader (512字节代码)做的事情
  - 做好系统设置、准备好堆栈
  - 把ELF Header读到某个位置(如何读磁盘？)
  - 把ELF文件加载到内存
    - 执行((void (*)())elfhdr->entry)();

操作系统是个C程序

- 此时，已经有了C程序执行的资源了
  - 操作系统(C程序)独占处理器
  - 中断处于关闭状态
  - 代码、数据、bss (均被bootloader加载和初始化)
  - 堆栈 (bootloader已设置)
  - 还有若干闲置的内存

- C程序就能执行了！
  - 不要高兴得太早：光有C还是不够的
  - 还需要通过指令与计算机硬件交互

AbstractMachine簡介 (1)

C程序的执行

- 操作系统作为C程序执行在CPU上
  - 操作系统(C程序)独占处理器
  - 中断处于关闭状态
  - 代码、数据、bss (均被bootloader加载和初始化)
  - 堆栈 (bootloader已设置)
  - 还有若干闲置的内存

- 但是没有库，它什么也干不了啊?
  - 假设禁止使用内嵌汇编访问硬件
  - 实现操作系统到底需要哪些机制?

困难

- 无论运行在怎样的硬件上，操作系统都是个C程序
  - MIPS32
  - RISC-V
  - x86 (qemu/nmeu)

- 但每个硬件平台都不一样
  - 各自有各自的启动方法
  - 有的有MMU (x86)，有的没有 (MIPS32可选)
  - 有些平台有多个处理器
  - ……

怎么在这些平台上实现操作系统?

- 回顾：操作系统是个C程序

- 支持I/O和进程/虚存抽象
  - 提供执行C程序的环境
  - 能让C程序访问I/O设备
  - 能调用中断/异常处理程序
  - 能实现虚拟存储
  - 能支持多个处理器

C程序的最基本API

- 提供两个API
  - void _putc(char ch); -- 打印一个字符
  - void _halt(int status); -- 停机
  - 以及一段可用的物理内存[start, end)

- 讨论：用这两个API我们可以做什么？

访问I/O设备的机制

- 每个I/O设备可以看作是若干控制寄存器
  - 读/写磁盘控制器、显存、时钟……
- I/O API
  - _Device *device(int n);
  - _Device: 包含id/name和read/write函数指针
    - size_t (*read)(uintptr_t reg, void *buf, size_t size);
    - size_t (*write)(uintptr_t reg, void *buf, size_t size);
- 讨论：有了I/O，我们可以做什么？

C程序 + I/O库 = 早期操作系统

- 你没有看错
  - 只要I/O库能输入/输入
  - 并且支持“切换到下一个程序”
  - 就是一个不错的批处理系统

响应中断/异常的机制

- 我们实际需要：
  - 能在中断/异常时准确获得发生的原因和所有的寄存器值
  - 能在中断/异常返回时选择指定的寄存器现场
  - 能创建寄存器现场
  - 能控制中断的开/关

- 有这些就能实现分时多任务了！

The AbstractMachine：实现OS的必要机制

应用程序：busybox, lua, …

NCC 编译器

Nanos
操作系统
(IOE,ASYE,[PTE],[MPE])

AM (full)

NEMU
x86模拟器(IOE)

MicroBench
基准程序

LiteNES
NES模拟器

Nalyze
代码分析工具

TRM + IOE + [ASYE] + [PTE] + [MPE]

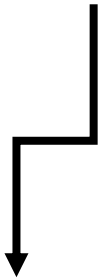
NPC
MIPS计算机

FPGA

Linux/QEMU

操作系统实验：机制与策略

- AM(硬件)提供了实现操作系统的必要机制



操作系统：使用这些机制的策略

- 操作系统提供运行应用程序的必要机制

- 这是构造大型软件的基本准则
  - 本质：人是有局限性的
  - 大脑容量是有限的，只能理解少量组件的交互