操作系统概述

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

操作系统课程简介

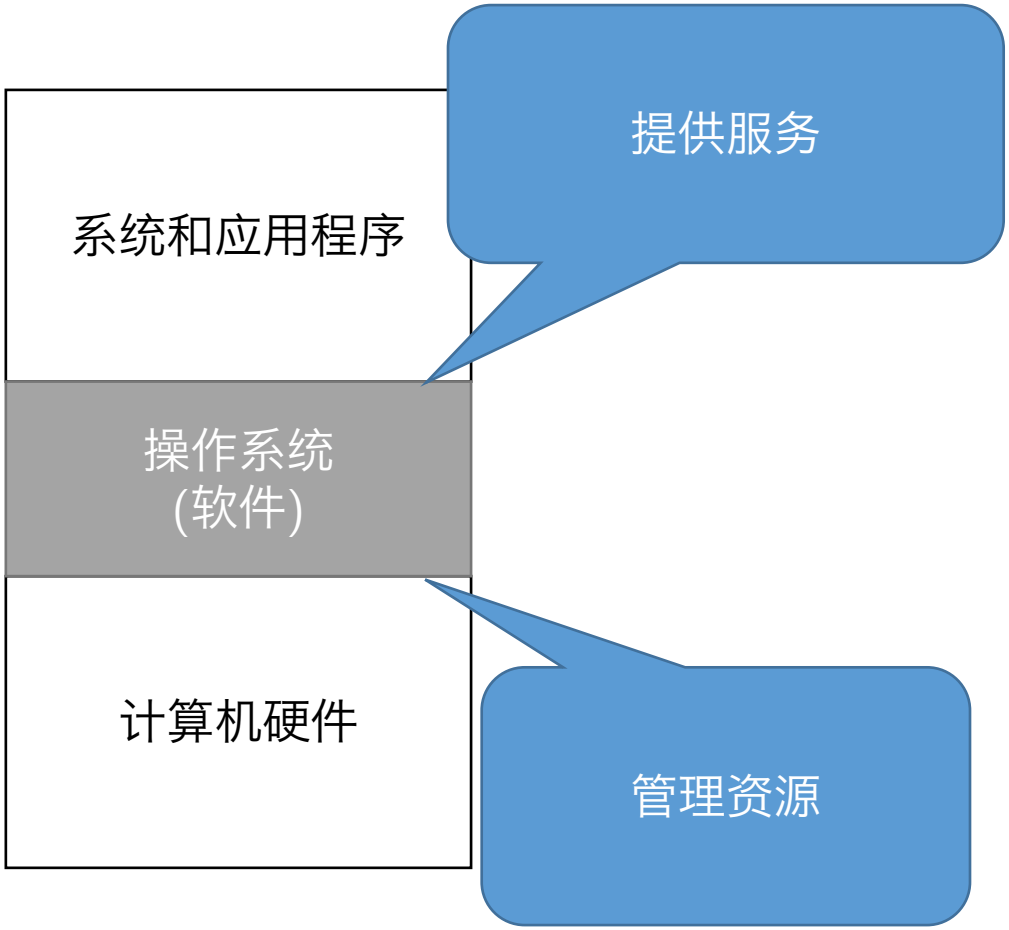
课程信息

- 操作系统：为什么？是什么？怎么造？
  - 课程主页：http://moon.nju.edu.cn/~jyywiki
  - 第一年的slides是匆忙赶工的，请不要公开传播
- 成绩
  - 期末考试40%, 期中测验10%, 系统实验25%, 编程实验25%
  - 出勤不影响成绩，但不来上课，吃亏的应该是你
  - 上课会请同学回答问题，但不作为任何评分依据
- Office Hours
  - 计算机楼809, 每周一14:00-16:00、每周四18:30-20:30
  - 改进意见/建议，欢迎来(或私下)讨论

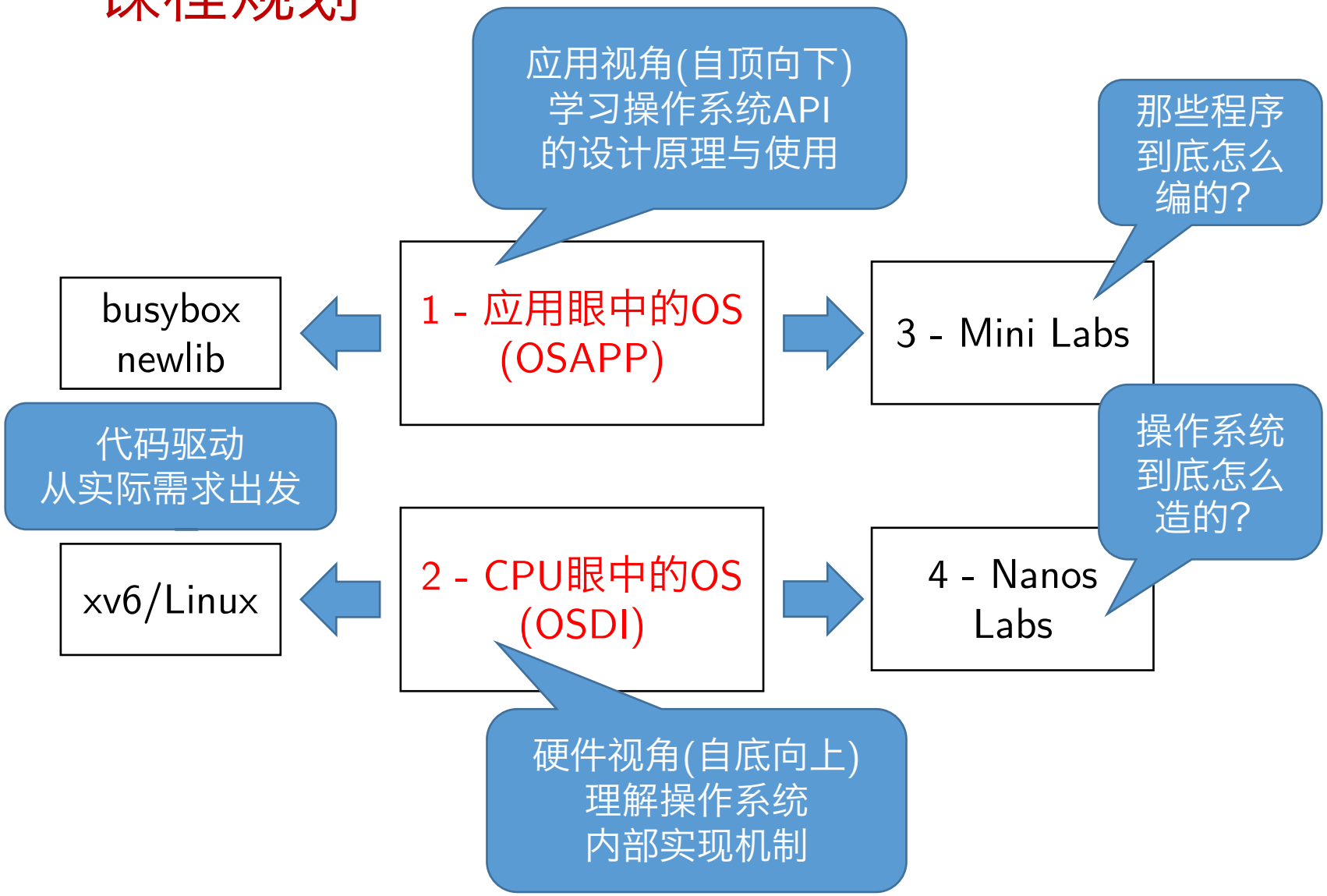
操作系统课程的位置

- ICS课程
  - 理解计算机系统栈的全貌
  - 理解ISA给上层软件提供的机制
    - 计算、I/O、中断、存储保护……
- 操作系统课程
  - 帮助你回答一个问题：如何利用ISA提供的功能，支撑起整个计算机软件栈？

操作系统概述



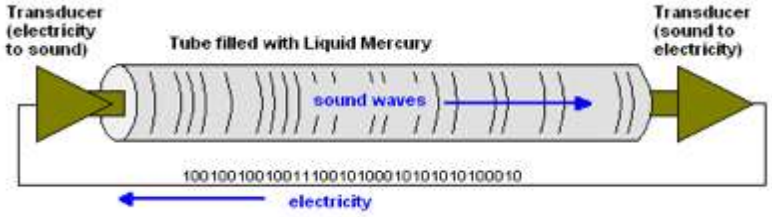
课程规划


应用视角(自顶向下)
学习操作系统API
的设计原理与使用
那些程序
到底怎么
编的?
busybox
newlib
1 - 应用眼中的OS
(OSAPP)
3 - Mini Labs
代码驱动
从实际需求出发
操作系统
到底怎么
造的?
xv6/Linux
2 - CPU眼中的OS
(OSDI)
4 - Nanos
Labs
硬件视角(自底向上)
理解操作系统
内部实现机制

为什么会有操作系统?
(操作系统的历史)

EDSAC (1949)

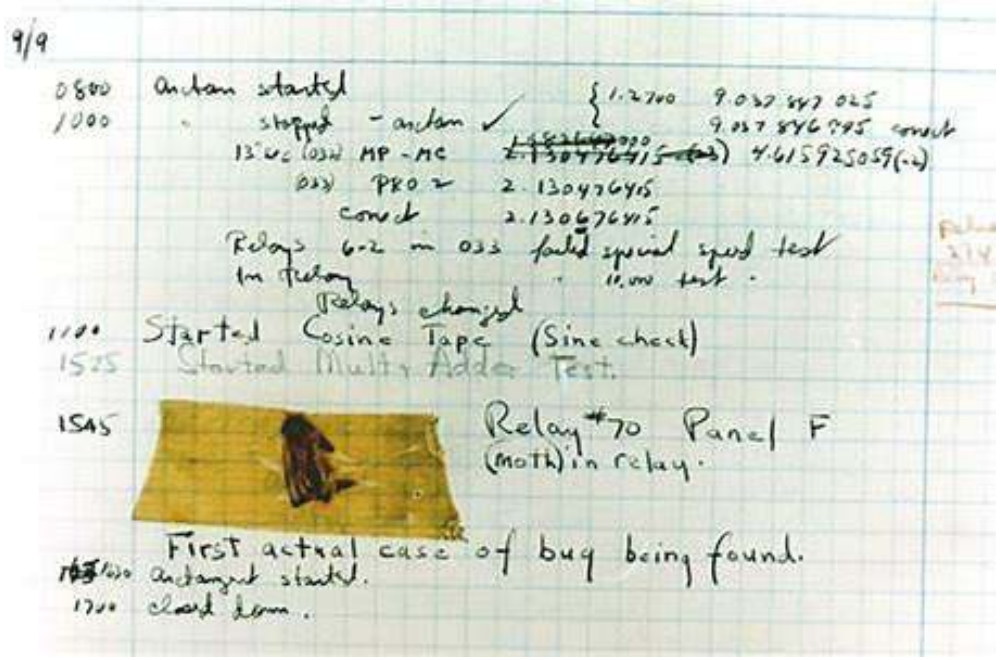
- 电子管、延迟线内存、RISC、打孔卡片、子程序
  - 运行的第一个程序：打印平方数/素数表



EDSAC: 36bits/line, 256 lines

1940s的操作系统

- 没听说过“操作系统”这么回事
- 大家还在de真的bug

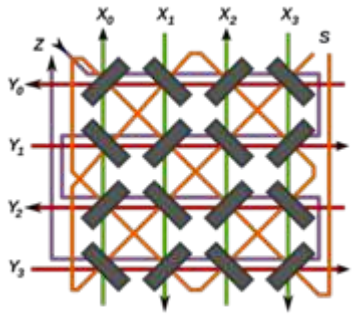


IBM 704 (1954)

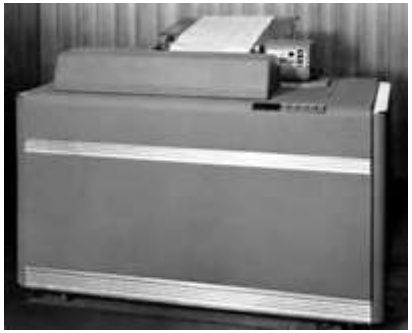
它需要怎样的操作系统?

Fortran代码

- 晶体管、打孔卡片、磁芯内存、中断



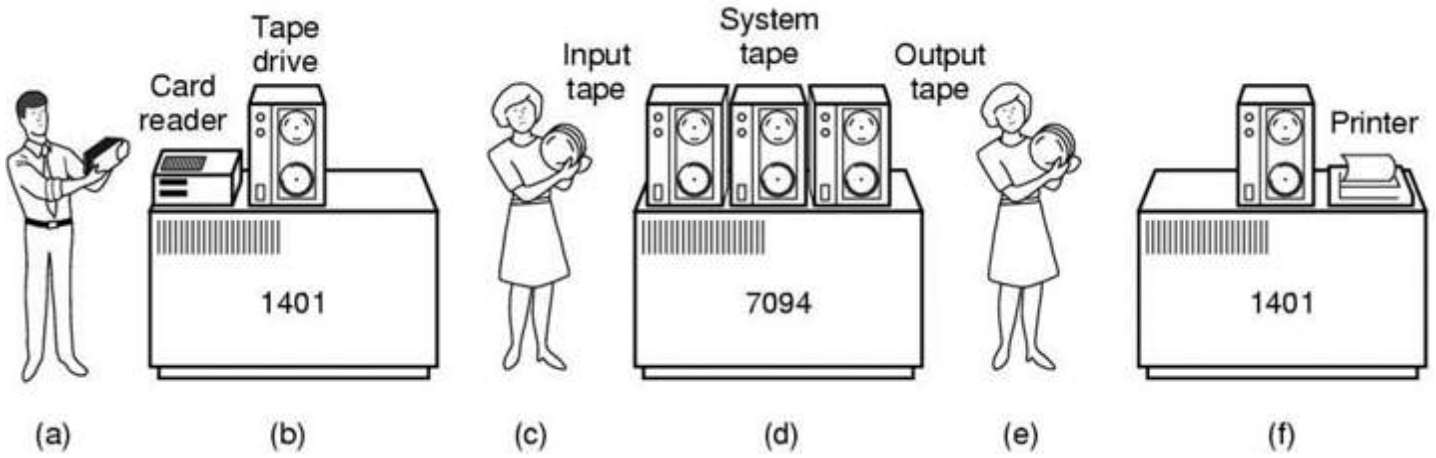
读卡器



打印机

1950s的操作系统

- 用来操作(operate)任务(jobs)的一系列子程序
  - 批处理系统，跑完一个任务以后跑下一个



PDP-1 (1959)

- 更快的计算机、终端、外设，各种脑洞大开的应用

它需要怎样的操作系统?

1960s的操作系统

- 多道程序系统
  - 系统里驻留多个程序(内存够大了)
  - 当一个程序得不到资源(等待龟速I/O设备)时，换另一个上
  - hack这些系统能实现分时共享
    - 不管你有没有资源，都换另一个上

PDP-11 (1970)

- 更快的存储器、内存映射I/O、优先级中断、(可选的)MMU、cache、更丰富的外设……与今天无异的应用

它们需要怎样的操作系统?

1970s+的操作系统

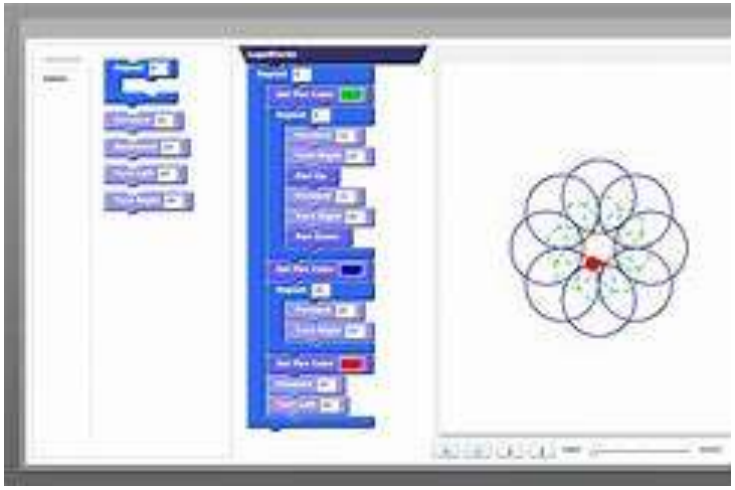
- 分时多任务现代操作系统
  - UNIX (1971), C、信号、管道、grep (1973), BSD socket (1983), procfs (1984), ...
  - 操作系统的基本功能趋于稳定
    - 虽然设计、实现有差别，但都能支持同样类型的应用

2010s的计算机

- 物联网
  - 嵌入式/物联网设备
- 个人计算
  - 智能手表、智能手机、个人电脑
- 数据中心(分布式计算系统)
  - 在PB量级数据上的计算和查询
  - 利用最新的计算/存储/网络技术

2010s的操作系统

- 大家越来越不需要了解什么是操作系统了
  - 爷爷奶奶也会玩微信了
  - 我们在steam、网页上的时间越来越长了
  - 小朋友们不用花很多时间去了解 “操作系统的使用和编程了”



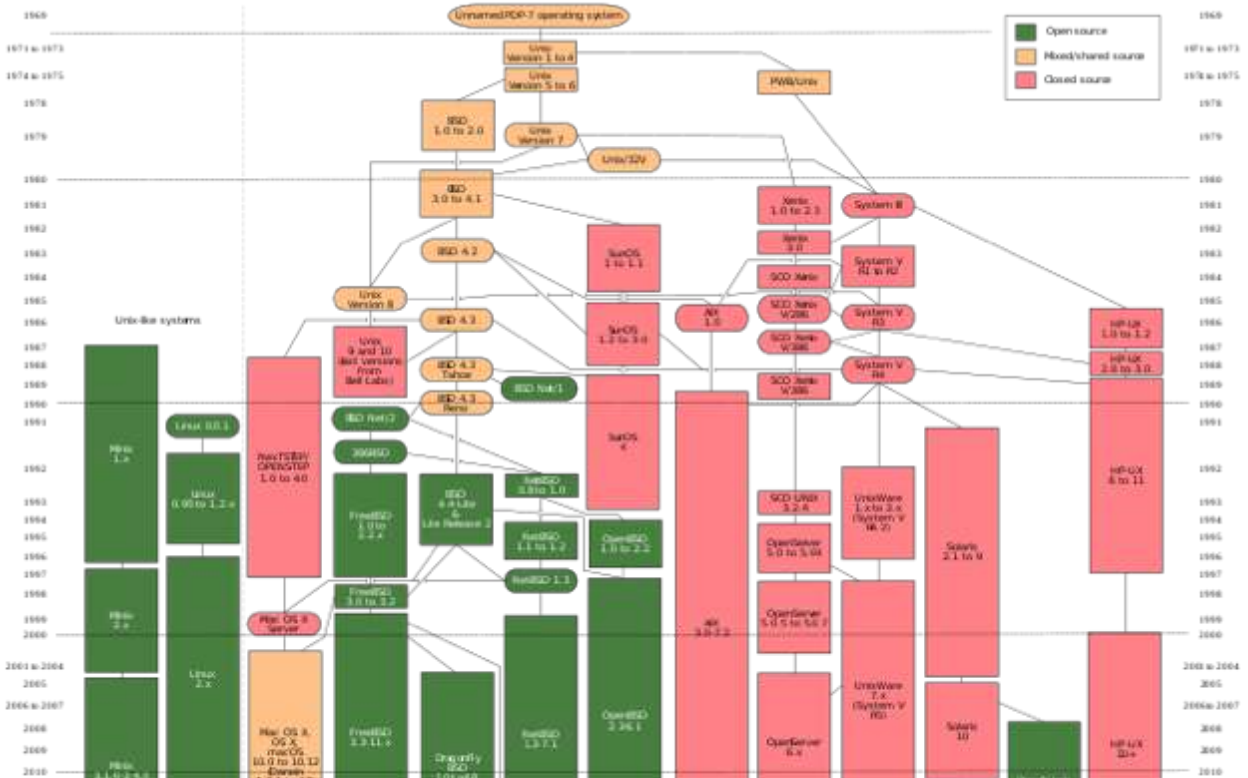
透过历史，我们看到了什么？

- 硬件、操作系统、应用是同步发展的
  - 计算机 → 计算应用 → 核弹爆炸了，人类登月了，想要更多应用
  - 丰富的I/O设备 → 中断、批处理系统 → 各种神奇应用
  - 硬件性能增强 → 多道程序 → 虚拟存储 → 分时共享 → UNIX
  - UNIX → 丰富的应用 → 自由软件 → Linux → 更丰富的应用
  - 巨大的应用数据 → 分布式系统 → 更多的应用

- 应用驱动了操作系统的发展
  - 时至今日，操作系统已经是计算机系统栈中非常稳固的一层
  - 从台前到幕后是其成熟的标志

还有呢?

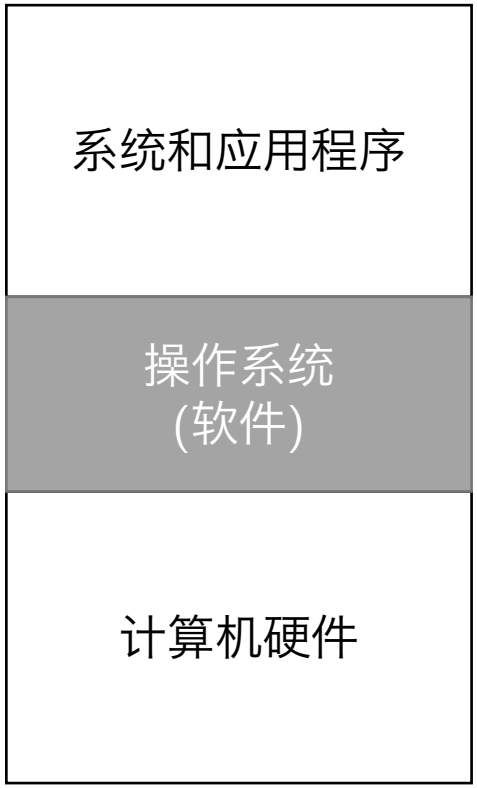
- 里程碑式的作品和硬件的发展终于成就了这一切：UNIX
  - 许多伟大的想法：C、shell、文件系统设计、管道、……
  - BSD, Linux, macOS都是UNIX的后代，Windows NT也参考了其设计，甚至一度包含POSIX subsystem



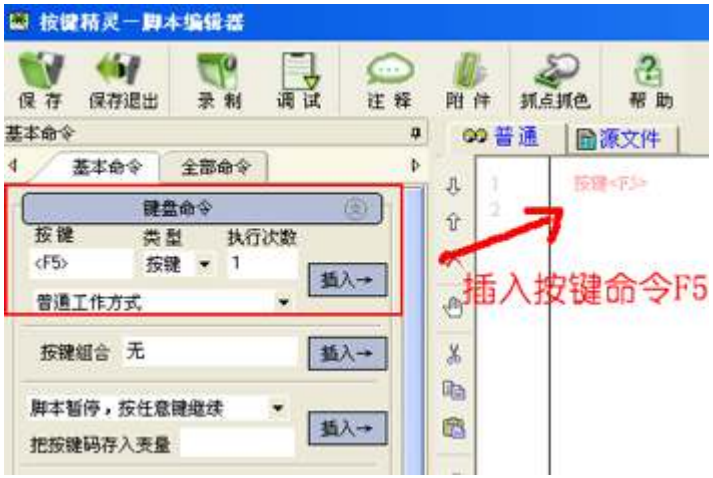
应用程序眼中的操作系统

什么是应用程序?

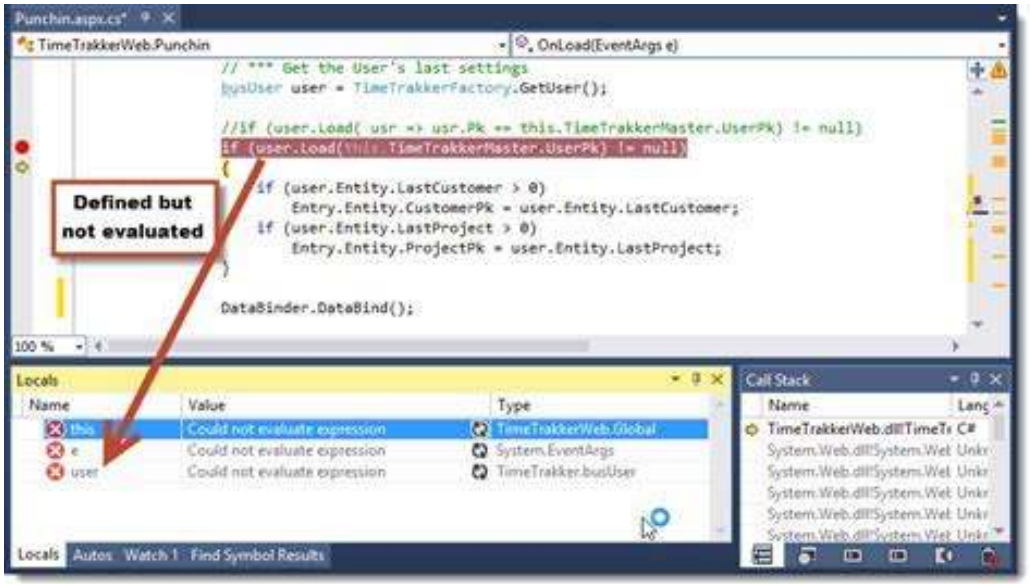
- Hello World
  - printf("Hello World\n");
- 编程课上学写的应用程序
  - scanf("%d%d", &a, &b); printf("%d\n", a + b);
- ICS课上学过的应用程序(nemu)
  - while (1) { exec_inst(); }
- 用户和操作系统交互的程序：shell、图形界面……
  - 它们属于系统软件程序，但系统/应用的界限本身就是模糊的



更多的应用(系统)程序


按键精灵一脚本编辑器
保存
保存退出
录制
调试
注释
附件
抓点抓色
帮助
基本命令
普通
源文件
基本命令
全部命令
键盘命令
按键
类型
执行次数
<F5>
按键
1
插入→
普通工作方式
按键组合
无
插入→
脚本暂停，按任意键继续
插入→
把按键码存入变量
插入按键命令F5


360
www.360.cn


Punchin.aspx.cs*
TimeTrakkerWeb.Punchin
OnLoad(EventArgs e)
// *** Get the User's last settings
busUser user = TimeTrakkerFactory.GetUser();
//if (user.Load(usr => usr.Pk == this.TimeTrakkerMaster.UserPk) != null)
if (user.Load(TimeTrakkerMaster.UserPk) != null)
{
if (user.Entity.LastCustomer > 0)
Entry.Entity.CustomerPk = user.Entity.LastCustomer;
if (user.Entity.LastProject > 0)
Entry.Entity.ProjectPk = user.Entity.LastProject;
}
DataBinder.DataBind();
Defined but not evaluated
Locals
Name
Value
Type
this
Could not evaluate expression
TimeTrakkerWeb.Global
e
Could not evaluate expression
System.EventArgs
user
Could not evaluate expression
TimeTrakker.busUser
Call Stack
Name
Locals
Autos
Watch 1
Find Symbol Results

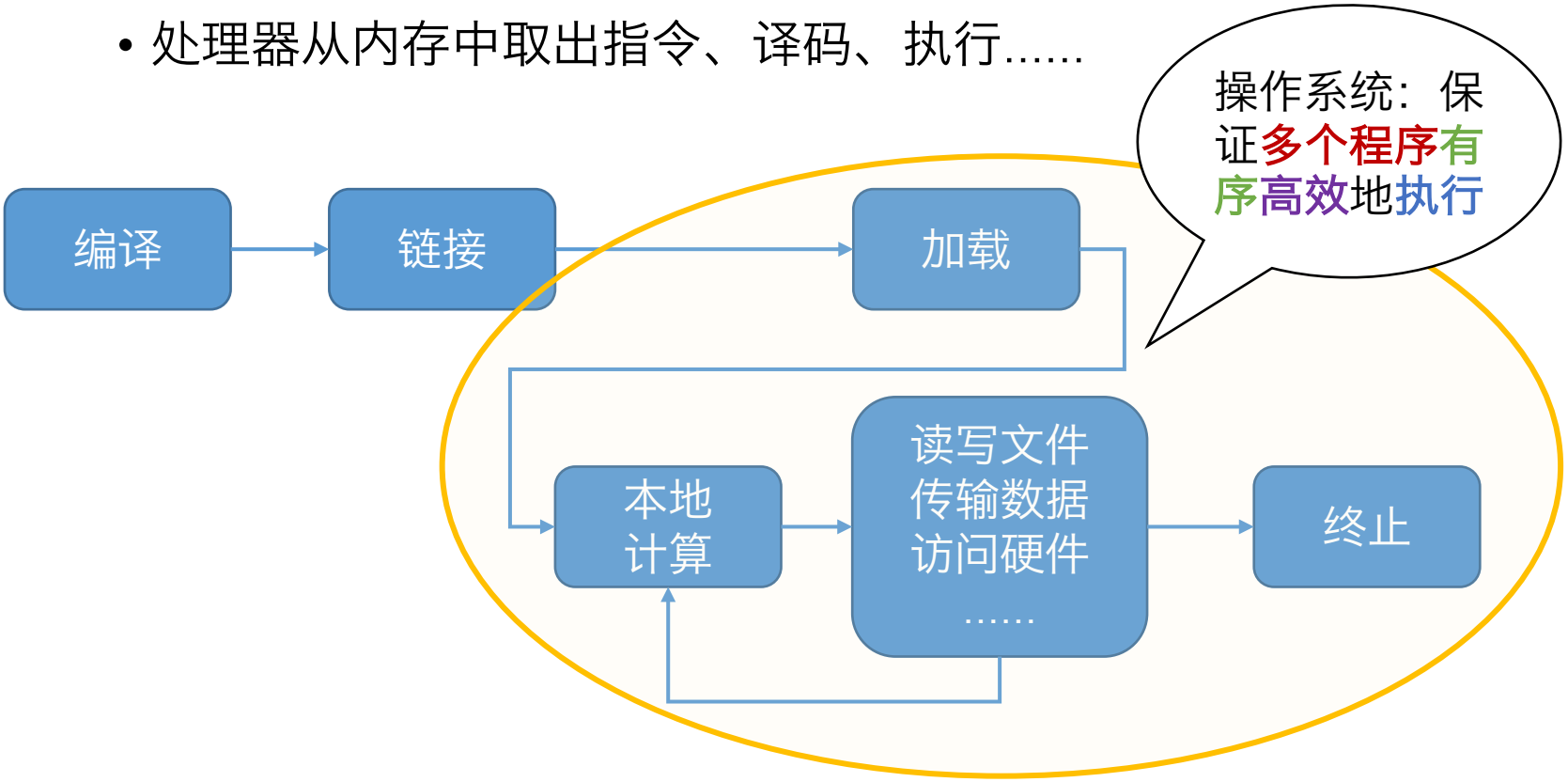
应用程序眼中的操作系统

- 在应用眼中，操作系统提供了程序的执行环境
  - 程序执行需要使用内存：代码、数据和堆栈
  - 程序执行需要消耗CPU的时间

- 同时提供了程序和操作系统世界中对象和资源交互的方式
  - 如果程序直接访问系统状态、I/O和中断，编程就太困难了
  - 操作系统提供足够的便利，能响应应用程序的各类合理请求
    - 发送用户界面事件
    - 监控系统行为
    - 获得系统运行时信息、管理其他运行的程序
    - 调试/改变其他程序行为

应用程序与操作系统

- 程序 = 指令的序列
  - 在ICS课程中，已经知道程序是如何在计算机中表示的
  - 处理器从内存中取出指令、译码、执行……



进入操作系统：Hello, OS World

- 一个真实的小程序在操作系统上执行，到底发生了什么？
  - 库函数做了什么？
  - 操作系统扮演了什么角色？
  - 操作系统内部发生了什么？


```
nt main(int argc, char *argv[]) {
  printf(“Hello World from %s”,
     argv[0]);
  return 0;
}
```

深入printf

- 问题
  - 为什么我用./a.out > a.txt就能输出到别的地方?
  - 为什么我用./a.out | grep Hello就能输出到另一个程序的输入?
  - printf到底是什么神奇的东西? printf的代码在哪里?

```
400526: 55                   push   %rbp
400527: 48 89 e5             mov    %rsp,%rbp
40052a: bf c4 05 40 00       mov    $0x4005c4,%edi
40052f: e8 cc fe ff ff       callq  400400 <puts@plt>
400534: b8 00 00 00 00       mov    $0x0,%eax
400539: 5d                   pop    %rbp
40053a: c3                   retq
```

libc中的printf

- 库函数中相当一部分工作是本地计算
  - 思考：如何实现sprintf(char *buf, const char *fmt, ...);?
- 但有一部分是程序本地计算永远无法完成的
  - 向屏幕输出字符
  - 写入磁盘
  - 关闭计算机
  - 读取系统中的其他进程
- 它们就是应用程序和操作系统之间的真正接口(系统调用)

讨论：目前操作系统无论其内部设计如何，应用程序都能实现printf或与之等价的功能

- 功能：将字符串打印(到哪里？)，等待打印完毕后返回

printf应该如何用本地计算和系统调用实现？