

# 读书笔记

## 操作系统：原理与实践

# ① 第一卷内核与进程

## 引言

### 内核抽象

# 引言

考虑一个 Web server，操作系统的部分工作是使得开发类似 web server 这样的应用更加容易。从 Web server 的角度，操作系统可能要提供一些支持，比如：

- Web server 需要访问文件或调用外部程序计算并获取结果 - 操作系统怎么让不同的应用互相通信？
- 客户端可以并发访问，为获得更好的体验，server 同时处理多个客户端的请求 - 操作系统如何支持应用同时做多件事情
- 随着业务增长，web server 可能要部署在其他硬件上 - 操作系统如何屏蔽硬件差异，以避免重写 web server？

# 操作系统定义

操作系统是一个软件层，管理计算机资源，为用户及其应用程序提供服务。

# 操作系统角色

操作系统作为硬件与应用的中间层，需要扮演三个角色：

- 裁判 (Referee)：资源共享。应用共享硬件资源，需要操作系统决定资源何时分配给哪个应用。
- 魔术师 (Illusionist)：提供一个物理硬件的抽象层，使得应用仿佛拥有完整的、资源不受限的计算机从而简化设计。例如应用不必关注物理内存分布、不必关注其他应用的资源竞争。
- 粘合剂 (Glue)：为应用程序提供公共服务或组件，例如文件系统、统一的用户界面观感、IO。

## 其他系统的类似角色

除了操作系统，很多其他系统也有类似的设计挑战，比如，云计算、浏览器、数据库系统、互联网。以云计算为例：

- Referee：资源在云上不同应用之间如何分配？
- Illusionist：云的资源一直在变化，应该如何向应用开发者隔离硬件变化？
- Glue：云上服务通常跨主机运行。云软件需要提供那些服务来协调不同的服务以及共享数据？

## 操作系统评估标准：

- 可靠性和可用性：系统是否按照预期运行及其运行时间比例。
- 安全性：是否能够防止恶意攻击及保护用户数据隐私。
- 可移植性：系统是否易于适配新的硬件。
- 性能：包括响应时间、吞吐量以及可预测性。
- 流行度：应用程序、硬件、用户生态。

# 操作系统：过去现在与未来

- 早期操作系统：操作系统主要是通过提供一些公共服务（如标准 IO 程序）来减少编程错误。
- 批处理操作系统
  - 依次对每个任务加载、执行、卸载的步骤。在一个任务执行时，利用 DMA，可以并行进行 IO 操作。
  - 多任务操作系统：允许多个任务并发执行，当一个任务需要 IO 给出 CPU 时，其他任务可以执行，提高了处理器利用率。也带来了更高的隔离需求。
  - 虚拟机最早用来解决批处理系统难以 Debug 的问题。
- 分时操作系统：支持交互式使用，而不只是批处理。
- 现代操作系统：随着计算机技术的发展，各种现代操作系统运行在不同的硬件上，包括个人电脑、智能手机、嵌入式设备、服务器等。
- 未来操作系统展望：
  - 安全性和可靠性的更高要求
  - 操作系统需要适应底层硬件发展。数据中心向大规模、异构方向发展；无处不在的便携硬件，如各种可穿戴设备。



# ① 第一卷内核与进程

引言

内核抽象

# 概述

- 操作系统的一个核心角色是保护——把 misbehaving 的程序和用户隔离开来，以避免影响其他应用或操作系统本身。
- 保护是实现某些操作系统其他目标的基础。如安全性、可靠性、隐私、公平资源分配。
- **进程是内核提供的受保护执行的抽象**
- 这种设计模式——可扩展的程序运行第三方脚本——在其他领域也有很多，比如浏览器。

# 进程抽象

- 进程是程序运行实例，
  - 执行指令
    - program counter
    - 不同类型的指令：算术、分支、访问 IO 等
  - 访问内存
    - 代码
    - 数据
    - 堆
    - 栈
- 操作系统通过进程控制块（PCB）管理和追踪进程
- 本章只讨论简单的单线程进程

## 进程保护

操作系统如何防止进程危害其他进程或者操作系统本身？

- 不考虑性能，一个假想的方案：由内核来执行用户进程的指令。
  - 对于用户态，在执行每个指令前检查其权限，如
    - 是否会跳转到其他进程内存？
    - 是否访问其他进程的数据？
- 如果想要用户态代码直接在处理器上运行，硬件至少需要提供以下支持：
  - 特权指令：只能在内核态执行，不能在用户态执行；例如
    - 用户态代码不能修改自己能访问的内存范围
    - 用户态代码不能关闭中断
  - 内存保护
    - 用户进程只能读写自己的内存，而不能访问其他进程或 kernel 的内存
    - 内存保护的一种简单方案：每个进程通过 base 和 bound 两个寄存器指定合法的内存地址。
    - 相比虚拟地址方案，缺少很多特性。
  - 时钟中断：允许内核周期性重获处理器的控制权，才有可能进行多任务调度。

# 模式切换

- 用户态到内核态
  - 外部中断
  - 处理器异常
  - 系统调用
- 内核态到用户态
  - 新建进程
  - 切换进程
  - 从中断、异常、系统调用中返回
  - Upcall(内核向用户态发起的调用): 如 linux 下的信号

# 实现安全的模式切换

- context switch 的实现至少应该满足：
  - 受限地进入内核：用户进程不能任意访问内核内存，硬件必须确保进入内核的入口点是由内核设置的入口点。
  - 处理器状态的原子切换：例如不能发生 PC 切换到内核态代码，但内存保护状态还没切换到内核态。
  - 中断处理对被中断的进程透明。
- 需要软硬件共同实现，相关机制包括：
  - 中断向量
  - Interrupt stack/Kernel stack：处理器和内核配合，在中断或系统调用时切换栈
  - 中断屏蔽：有些内核代码执行过程不能被中断，需要屏蔽中断
  - 保存/恢复被中断的现场，主要是寄存器。

# 具体实现

- x86 中断处理的模式切换
- 系统调用的实现
- 创建新进程
- 实现 upcall

# Case Study

- 从 BIOS 到 bootloader 到 Kernel 启动过程
- 虚拟机