第 2 讲：中断、异常与系统调用

第一节：从 OS 角度看计算机系统

向勇、陈渝、李国良

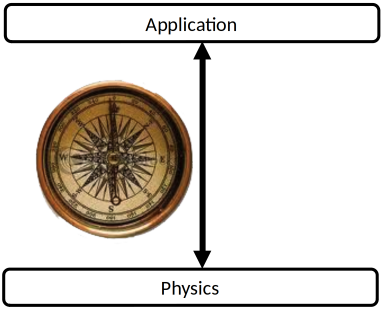
清华大学计算机系

xyong,yuchen,liguoliang@tsinghua.edu.cn

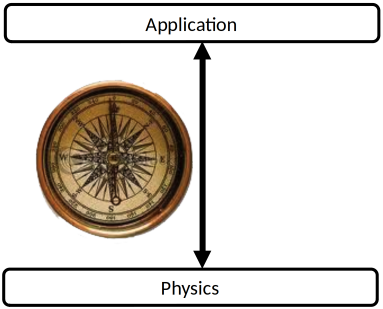
2021 年 2 月 24 日

提纲

计算机系统 [CS−152Berkeley]

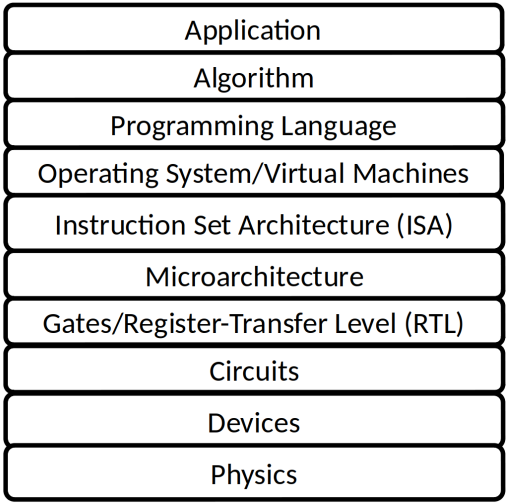


计算机系统 [CS−152Berkeley]

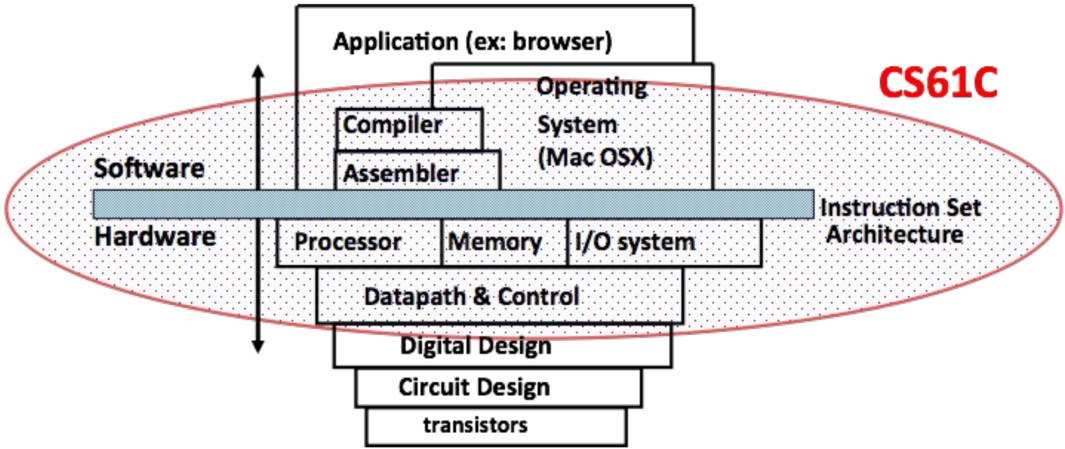


广义的定义, 计算机系统（computer architecture）是一种抽象层次的设计，用于实现可有效使用现有制造技术的信息处理应用。[CS−152Berkeley]

计算机系统的抽象层次 [CS−152Berkeley]



软硬件接口 [CS−152Berkeley]



提纲

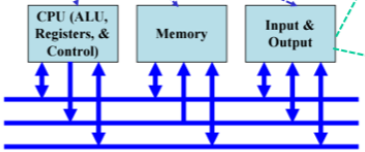
1. 第一节：从 OS 角度看计算机系统
   - 总体视图
   - OS 与体系结构的关系
   - OS 与应用程序的关系
   - 隔离：概述
   - 隔离：虚拟内存
   - 隔离：特权模式
   - 隔离：中断机制

OS 与体系结构的关系

Kernel

- Process
- Memory
- Device
- File System
- …

- Memory
- ALU
- Control Unit
- I/O System



再看计算机系统

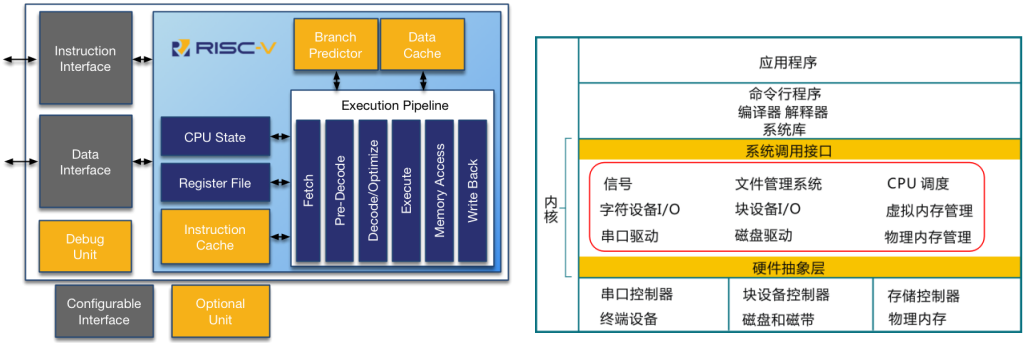
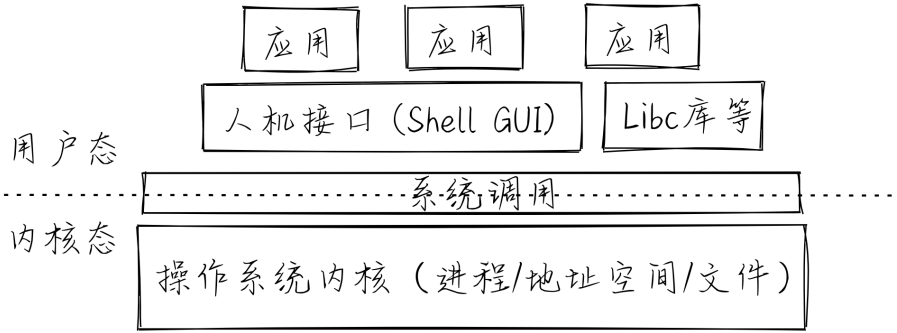


图: 再看计算机系统 – 从 OS 角度

提纲

1. 第一节：从 OS 角度看计算机系统
   - 总体视图
   - OS 与体系结构的关系
   - OS 与应用程序的关系
   - 隔离：概述
   - 隔离：虚拟内存
   - 隔离：特权模式
   - 隔离：中断机制

OS 与应用程序的关系



隔离：程序调用

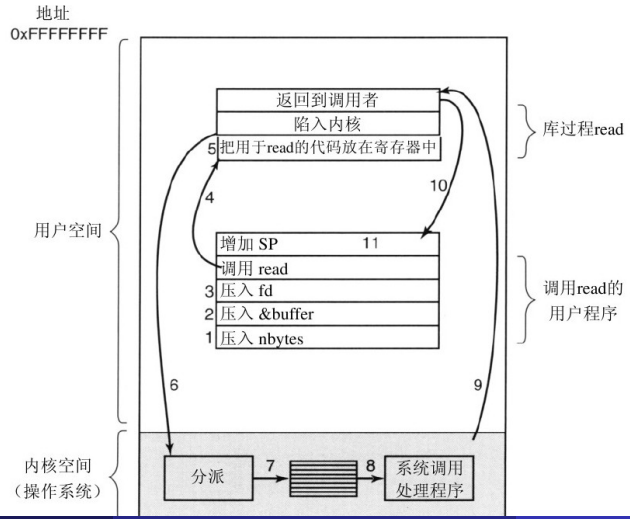
- 程序调用 *ssize_t read(int fd, void *buf, size_t count);* 会发生什么？
- 我们可以在应用程序中直接调用内核的函数吗？
- 我们可以在内核中使用应用程序普通的函数调用吗？

隔离：程序调用

- 程序调用 *ssize_t read(int fd, void *buf, size_t count);* 会发生什么?
- 我们可以在应用程序中直接调用内核的函数吗?
- 我们可以在内核中使用应用程序普通的函数调用吗?
- 程序调用的特征
  - 好处：执行很快;
  - 好处：灵活-易于传递和返回复杂数据类型;
  - 好处：程序员熟悉的机制,...
  - 坏处：应用程序不可靠，可能有恶意，有崩溃的风险

OS 与应用程序的关系

*ssize_t read(int fd, void *buf, size_t count);*



OS 与应用程序的关系

进 程 管 理

调　　用	说　　明
pid = fork()	创建与父进程相同的子进程
pid = waitpid(pid, &statloc,options)	等待一个子进程终止
s = execve(name, argv, environp)	替换一个进程的核心映像
exit(status)	中止进程执行并返回状态

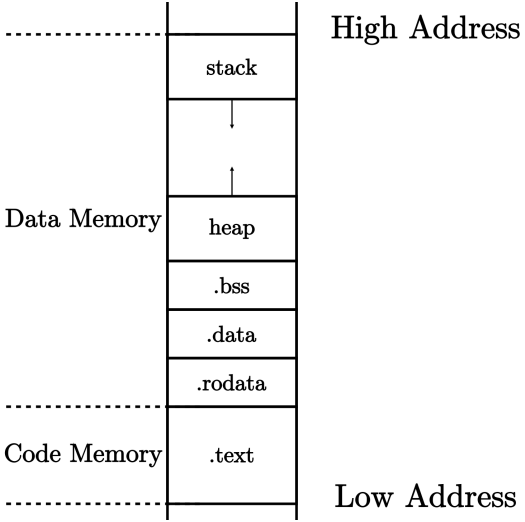
OS 与应用程序的关系

文件管理

调　用	说　明
fd = open(file, how, ...)	打开一个文件供读、写或两者
s = close(fd)	关闭一个打开的文件
n = read(fd, buffer, nbytes)	把数据从一个文件读到缓冲区中
n = write(fd, buffer, nbytes)	把数据从缓冲区写到一个文件中
position = lseek(fd, offset, whence)	移动文件指针
s = stat(name, &buf)	取得文件的状态信息

OS 与应用程序的关系

Q: 在内存方面：OS 与应用程序的关系？



提纲

隔离： 什么是隔离?

- 强制隔离以避免对整个系统的可用性/可靠性/安全影响
- 运行的程序通常是是隔离的单元

隔离： 什么是隔离?

- 强制隔离以避免对整个系统的可用性/可靠性/安全影响
- 运行的程序通常是是隔离的单元
- 防止程序 X 破坏或监视程序 Y
  - 读/写内存，使用 100%的 CPU，更改文件描述符
- 防止进程干扰操作系统
- 防止恶意程序、病毒、木马和 bug
  - 错误的过程可能会试图欺骗硬件或内核

隔离： 主要的隔离方法?

- 地址空间 address spaces
  - 一个程序仅寻址其自己的内存
  - 每个程序若无许可，则无法访问不属于自己的内存

隔离： 主要的隔离方法?

- 地址空间 address spaces
  - 一个程序仅寻址其自己的内存
  - 每个程序若无许可，则无法访问不属于自己的内存
- CPU 硬件中的特权模式/中断机制
  - 防止应用程序访问设备和敏感的 CPU 寄存器
  - 例如地址空间配置寄存器
  - 例如打断一直占用 CPU 的应用程序

提纲

1. 第一节：从 OS 角度看计算机系统
   - 总体视图
   - OS 与体系结构的关系
   - OS 与应用程序的关系
   - 隔离：概述
   - 隔离：虚拟内存
   - 隔离：特权模式
   - 隔离：中断机制

虚拟内存

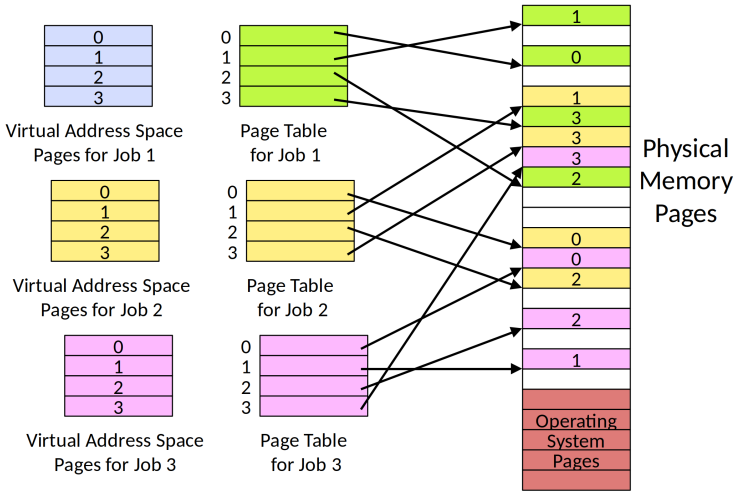

0
1
2
3
Virtual Address Space
Pages for Job 1
0
1
2
3
Page Table
for Job 1
0
1
2
3
Virtual Address Space
Pages for Job 2
0
1
2
3
Page Table
for Job 2
0
1
2
3
Virtual Address Space
Pages for Job 3
0
1
2
3
Page Table
for Job 3
1
0
1
3
3
3
2
0
0
2
2
1
Operating
System
Pages
Physical
Memory
Pages

图: 虚拟内存

虚拟内存

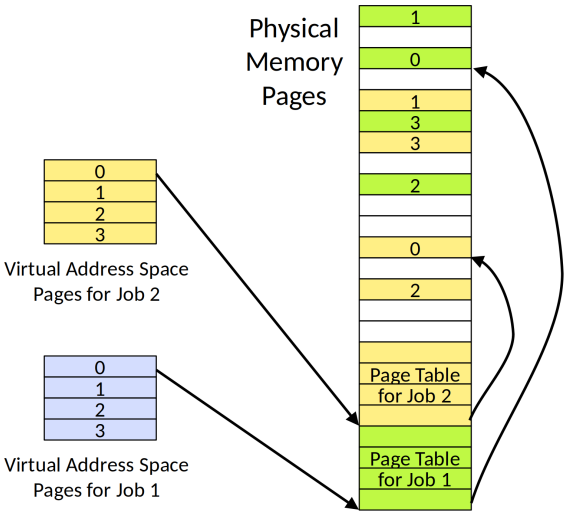

Physical
Memory
Pages
1
0
1
3
3
2
0
2
Page Table
for Job 2
Page Table
for Job 1
0
1
2
3
Virtual Address Space
Pages for Job 2
0
1
2
3
Virtual Address Space
Pages for Job 1

图: 页表

虚拟内存

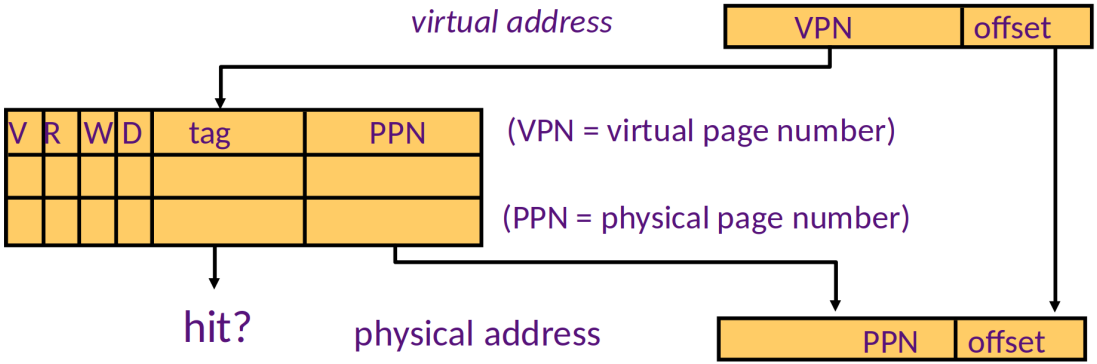

virtual address
VPN
offset
V
R
W
D
tag
PPN
(VPN = virtual page number)
(PPN = physical page number)
hit?
physical address
PPN
offset

图: TLB

虚拟内存

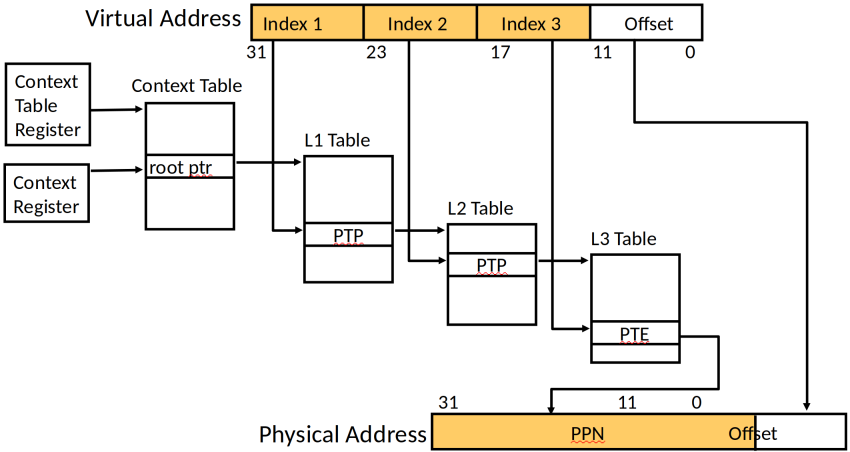


图: MMU 处理 TLB Missing

虚拟内存

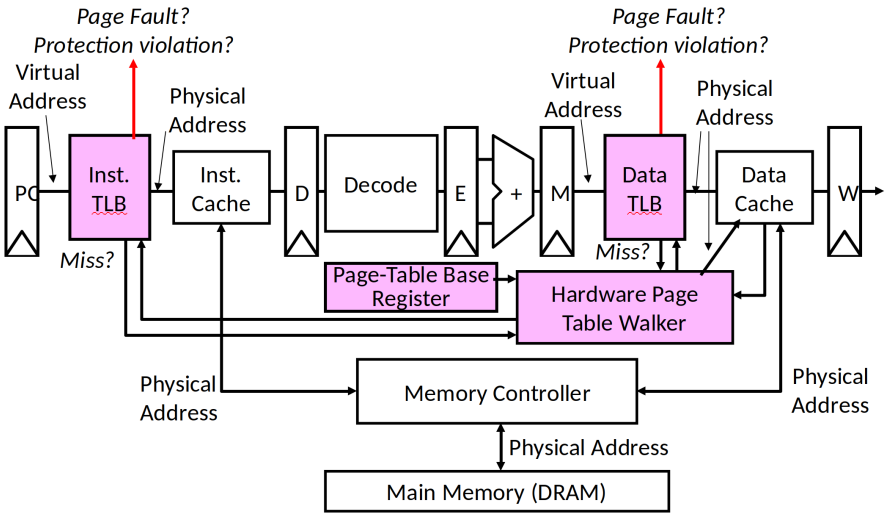


图: 带 MMU/TLB 的计算机系统

提纲

1. 第一节：从 OS 角度看计算机系统
   - 总体视图
   - OS 与体系结构的关系
   - OS 与应用程序的关系
   - 隔离：概述
   - 隔离：虚拟内存
   - 隔离：特权模式
   - 隔离：中断机制

特权模式

- CPU 硬件支持不同的特权模式
  - **Kernel Mode** vs **User Mode**
  - **Kernel Mode** 可以执行 **User Mode** 无法执行的特权操作
    - 访问外设
    - 配置地址空间（虚拟内存）
    - 读/写特殊系统级寄存器
  - OS kernel 运行在 **Kernel Mode**
  - 应用程序运行在 **User Mode**
  - 每个重要的微处理器都有类似的用户/内核模式标志

提纲

1. 第一节：从 OS 角度看计算机系统
   - 总体视图
   - OS 与体系结构的关系
   - OS 与应用程序的关系
   - 隔离：概述
   - 隔离：虚拟内存
   - 隔离：特权模式
   - 隔离：中断机制

中断机制

- CPU 硬件支持中断/异常的处理
- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
  - 硬件中断不是由任何一条专门的 CPU 指令造成，从这个意义上它是异步的。

中断机制

- CPU 硬件支持中断/异常的处理
- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
  - 硬件中断不是由任何一条专门的 CPU 指令造成，从这个意义上它是异步的。
- 硬件中断的异常处理程序通常称为中断处理程序（interrupt handle）
  - I/O 设备通过向处理器芯片的一个引脚发信号，并将异常号放到系统总线上，以触发中断；
  - 在当前指令执行完后，处理器从系统总线读取异常号，保存现场，切换到 Kernel Mode；
  - 调用中断处理程序，当中断处理程序完成后，它将控制返回给下一条本来要执行的指令。

中断机制

- CPU 硬件支持中断/异常的处理
- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
  - 硬件中断不是由任何一条专门的 CPU 指令造成，从这个意义上它是异步的。
- 硬件中断的异常处理程序通常称为中断处理程序（interrupt handle）
  - I/O 设备通过向处理器芯片的一个引脚发信号，并将异常号放到系统总线上，以触发中断；
  - 在当前指令执行完后，处理器从系统总线读取异常号，保存现场，切换到 Kernel Mode；
  - 调用中断处理程序，当中断处理程序完成后，它将控制返回给下一条本来要执行的指令。
- Timer 可以稳定定时地产生中断
  - 防止应用程序死占着 CPU 不放
  - 让 OS kernel 能周期性地进行资源管理

小结

- 了解计算机硬件与软件的接口
- 了解操作系统与应用程序的接口
- 了解操作系统如何隔离与限制应用程序