



## 第2章 进程与线程

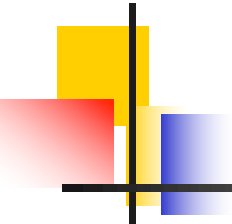
# 2.3 OS中的中断处理





## 2.3 OS中的中断处理

- OS是一堆功能各异的代码组合，在不同中断（事件）的驱动下，完成硬件资源的操作和管理
- 典型中断来源
  - 来自硬件设备的处理请求
  - 程序运行产生异常事件
    - 出错
    - 程序请求操作系统服务

- 
- 当事件发生时，处理器通常会立即终止当前任务，触发状态切换
  - OS为每一类事件定义了一组响应程序
    - 中断处理
    - 异常处理
  - 状态切换的途径
    - 均通过中断机制发生
    - 中断是用户态到核心态转换的仅有途径

## 2.3.1 中断的概念

- **中断**是指程序执行过程中，当发生某个事件时，中止CPU上现行政程序的运行，引出处理该事件的程序执行的过程。
- 中断通常也定义为一个能够改变CPU执行的指令序列的事件；OS是由中断（事件）驱动的
- 在物理上，中断与CPU内外部硬件电路产生的电信号相关

## 2.3.2 中断分类

- 同步中断：指令执行时由**CPU**控制单元产生的
  - 只有在一指令执行终止后**CPU**才会发出中断
- 异步中断：由其它硬件设备依据**CPU**时钟信号随机产生的
- 在**Intel** 处理器手册中，同步中断称为异常，异步中断称为中断

### 注：

- 中断的分类与处理器的体系结构有关、IBM中大型机、Intel微处理器等手册中对中断的分类均不相同；
- 从中断的产生及处理机理上，大致相同；
- 通常，中断和异常会统称为中断



# 中断分类

---

- 中断由定时器和I/O设备产生
- 异常由程序错误产生，或者由内核必须处理的异常条件产生
  - 更细粒度的分类
    - 异常：由程序错误产生的异常
    - 系统调用：请求系统服务

**中断、异常、系统调用三者有什么区别？**



## 2.3.3 中断、异常和系统调用

- 中断和异常希望解决的问题
  - 当外设连接计算机时，会出现什么现象？（中断）
  - 当应用程序出现非预期行为时，会出现什么现象？（异常）
- 系统调用希望解决的问题
  - 用户应用程序是如何得到系统服务？

# 中断、异常和系统调用

## ■ 产生原因

- 中断：来自硬件设备的处理请求

- 异常

  - 非法指令或其它原因导致当前指令执行失效

  - 如内存错误、除零错误等的处理请求

- 系统调用

  - 应用程序主动地向操作系统发生的服务请求

  - 是正在运行的程序所期待的事件，是由执行“访管指令”所引起的。

尽管产生的原因并不相同，但是在实现机制上，系统调用通常通过一种特殊的异常来实现；同时，异常与中断通常又都是通过相同的中断机制来实现





# 中断和异常的区别

## ■ 中断：

- 由与现行指令无关的中断信号触发的(异步的),
- 中断的发生与**CPU**处在用户模式或内核模式无关  
(系统不能确定中断发生的时间)
- 一般来说, 中断处理程序提供的服务不是为当前进程所需的;

## ■ 异常：

- 由处理器正在执行现行指令而引起的,
- 异常处理程序提供的服务是为当前进程所用的。
- 异常包括很多方面, 有出错(**fault**), 也有陷入(**trap**), 终止和编程异常等。



# 出错（Fault）和陷入(Trap)的区别

- 发生时保存的返回指令地址不同
  - **Fault**, 出错: 保存指向触发异常的那条指令
  - **Trap**, 陷入: 保存指向触发异常的那条指令的下一条指令。
- 从异常返回时
  - **Fault**会重新执行那条指令
  - **Trap**则不会重新执行那条指令。
  - 如缺页异常是一种出错, 而陷入可以应用在调试中。



# 异常和系统调用的区别

- 异常通常由意想不到的行为触发
- 系统调用则为主动请求系统服务
- Linux中异常
  - 处理器可探测的异常
    - 出错**fault**: 一般性错误, 可以修复
    - 陷入**trap**: 用于调试
    - 停止**abort**: 致命的、不可恢复错误
  - 编程异常**programmed exception**
    - 系统调用, 进程自愿进入核心



# 异常及异常处理

## ■ (1) 由硬件检测并处理的异常

### ■ 典型错误

- Page Fault
- 不合法的内存访问
- 除0

## ■ (2) 硬件自身出错产生的异常

- 通常在检测错误后会重新执行该指令
- PC、regs, mode等运行状态会由CPU保存



# 异常及异常处理

- (3) 有些异常的处理是修复异常，并返回上下文重新执行该指令
  - 比如Page Fault
    - OS会调用页面置换程序，修复页面错误
    - 恢复上下文，重新执行出错的指令
- (4) 有些异常是由OS通知相应进程
  - 比如Unix 系统中的信号
  - 直接“杀死”相应进程



# 异常及异常处理

---

- 如果异常发生在内核怎么办？
  - 空指针引用
  - 除零
  - 非法指令
- 此类严重错误会直接导致**OS崩溃**
  - 蓝屏





# 系统调用及其处理

- 系统调用时提供给用户态程序执行特权操作的接口
- 系统调用的响应
  - 触发事件，切换至内核态
  - 传递系统调用的参数
  - 保存状态以便完成系统调用后恢复执行
- 形如： `Int 0x80/sysenter`





# 系统调用及其处理

---

- 系统调用的参数传递
  - 系统调用号，保存在**eax**中
  - 其它的参数传递
    - **ebx, ecx, edx, esi, edi, ebp**





## 2.3.4 中断及中断处理

---

- 中断（异步中断）
  - 定时器
  - I/O 硬件中断装置





# 定时器

---

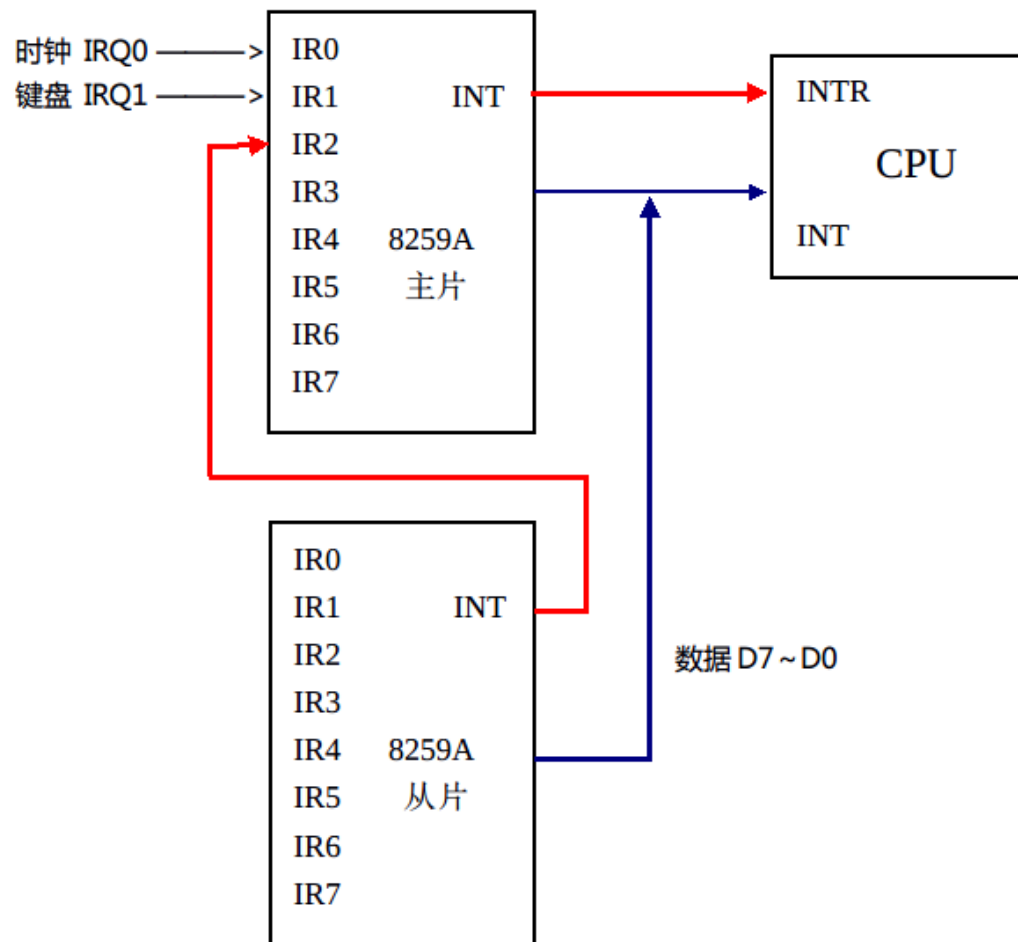
- OS回收控制的重要方式
  - 定时器会定时产生中断
  - 设定定时器是特权操作
  - 分时OS的调度
- 能够避免死循环
  - 程序出错或恶意程序的非法独占CPU
- 在时间相关的函数中被广泛使用 (e.g., `sleep()`)

# 中断装置

- 发现中断源并产生中断的**硬件**称中断装置
- 所有计算机系统都采用硬件和软件结合的方法实现中断处理
- 中断装置主要做以下三件事：
  - 发现中断源：当发现多个中断源，按照中断优先级别来先后响应
  - 保护现场：暂停当前程序运行，将中断点的**PSW**保存至核心栈，使得处理程序可运行
  - 启动处理中断事件的程序

# 中断装置

## ■ 可编程中断控制器8259A



# 中断装置

## ■ CPU响应中断的大致过程

- 检测到中断或异常后，自动保存CPU状态
  - 依据TR(Task Register)，定位当前进程的内存空间
  - 将CPU状态保存在相应内存地址
- 依据IDTR寄存器，定位到IDT内存地址
- 依据中断向量号在IDT中检索对应的中断处理程序
- 跳转至中断处理程序

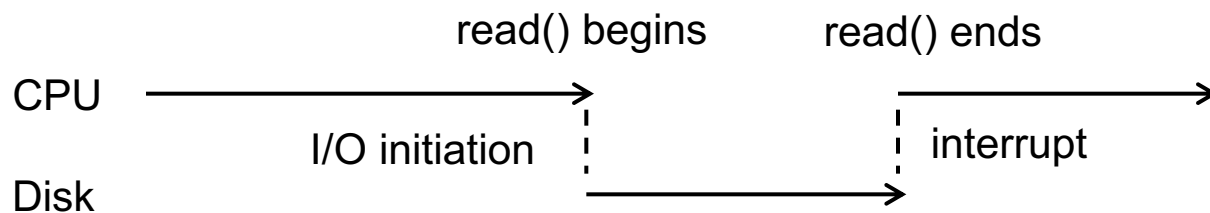
# 中断处理程序

- 处理中断事件的程序称为中断处理程序。它的主要任务是处理中断事件和恢复正常操作
- 不同中断源对应不同中断处理程序，故**快速找到中断处理程序的入口地址**是一个关键问题
- 中断处理程序主要做四项工作：
  - 保护未被硬件保护的一些必需的处理状态
    - CPU自动保存的只有EIP/EFLAGS/ESP/SS/CS
  - 识别各个中断源，分析产生中断的原因
  - 处理发生的中断事件
  - 恢复正常操作

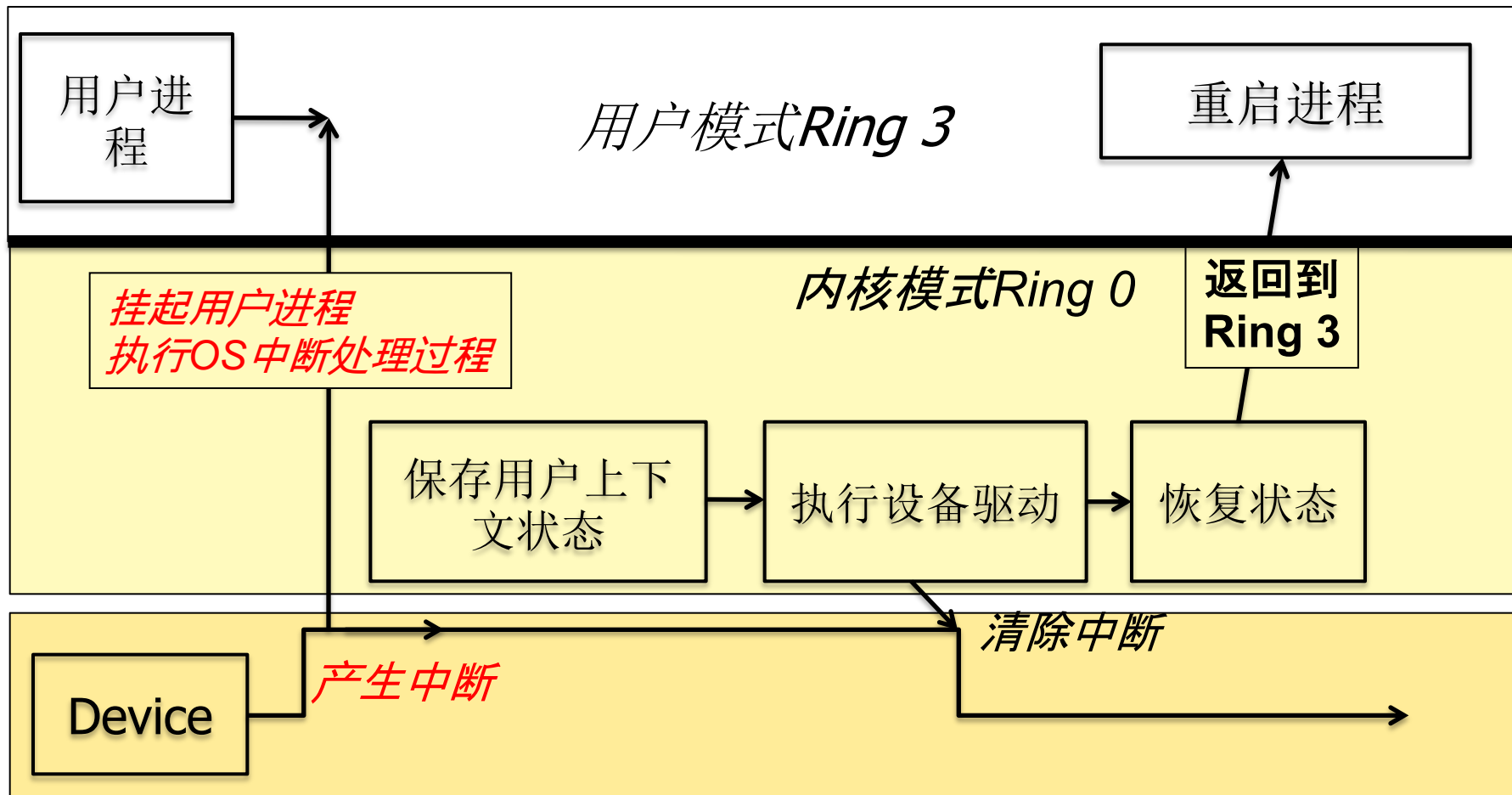


# I/O中断及其处理

- 中断是异步I/O的基础
  - OS 启动I/O
  - I/O设备开始工作
  - 完成后通过中断信号通知CPU
  - OS通过中断向量表来处理中断



# Interrupt的处理示例







# 同步与原子性

- 中断随时发生，并会打断当前的任务执行
- OS必须保证中断前后的同步
- 必须保证现场保护的原子性
  - 关中断以保证指令序列的原子性
  - 原子指令，如读/写内存





# 小结

---

- OS能获得的典型硬件支持

- 保护：保护硬件资源的受限访问

- 双模式（内核态和用户态）

- CPU提供的特权指令和非特权指令

- 内存保护（硬件支持的分段和分页）

- 支持并响应计算机系统的各类软硬件事件，如设备变化、程序的出错等异常

- 中断和异常机制

- 系统调用

- 定时器

- 支持多任务的并发与调度

- 中断发生时的状态切换

- I/O的控制

- 同步与互斥



# 小结

OS提供的服务	硬件特征
硬件资源保护和操作系统保护	双模式（内核态和用户态） 特权指令和非特权指令 内存保护
并发时的中断	中断向量
系统调用	可编程异常指令
I/O设备管理	I/O中断
任务调度	定时器
同步	原子指令





# 讨论

---

- 如果你需要设计一个支持多任务的实时操作系统，你会怎么利用这些硬件特性来达到要求？