1.3 运行环境_中断_系统调用

处理机运行模式

- 内核程序 VS 应用程序
  - 一种是操作系统内核程序
    - 操作系统的内核程序是系统的管理者，既可以执行特权指令，也可以执行非特权指令，运行在核心态
  - 另一种是用户自编程序：“应用程序”
    - 为了保证系统能安全运行，普通应用程序只能执行非特权指令，运行在用户态
- 内核
  - 有很多内核程序组成了【操作系统内核】，或简称内核
  - 内核时操作系统最重要最核心的部分，也是最接近硬件的部分
  - 甚至可以说，一个操作系统只要有内核就够了（eg：Docker—>仅需Linux内核）
  - 操作系统的功能未必都在内核中，如图形化用户界面GUI
- 特权指令 VS 非特权指令
  - 1）特权指令
    - 只能在内核态
    - 是指不允许用户直接使用的指令，如I/O指令、置中断指令，存取用于内存保护的寄存器、送程序状态字寄存器等的指令
    - 总结
      - 1. 开关中断指令
      - 2. 【写】【置】时钟指令
      - 3. 输入/输出指令（ I/O指令）
        - 控制IO设备
      - 4. 写PSW指令
      - **5. 中断返回指令**
      - 也是寄存器
  - 2）非特权指令
    - 是指允许用户直接使用的指令，不能直接访问系统中的软硬件资源
      - **仅限于访问用户的地址空间，不可访问内核空间**
      - 这也是为了防止用户程序对系统造成破坏
    - 总结
      - **1. 取数指令**
      - 2. 【读】时钟指令
      - 3. +-*/等运算指令
      - **4.（通用）寄存器清零指令**
        - 默认通用reg
        - **寄存器R0的内容取非**
      - **5. 压栈弹栈指令（push/pop）**
      - 6. 跳转指令（转移指令）
      - 7. 访管指令（陷入指令）trap指令
      - **8. 数据传送指令mov**
      - **9. 设置断点指令**
      - 【2021】
      - 用户可直接使用
  - 在CPU设计和生产的时候就划分了特权指令和非特权指令，因此CPU执行一条指令前就能判断出其类型
- 内核态 VS 用户态
  - 处于内核态 = 核心态 = 管态
    - 说明此时正在运行的是内核程序，此时可以执行特权指令
  - 处于用户态 = 目态
    - 说明此时正在运行的时应用程序，此时只能执行非特权指令
  - CPU中有一个寄存器叫 程序状态字寄存器（PSW），其中有个二进制位，1表示“内核态”，0表示“用户态”
- 内核态、用户态的切换
  - 内核态 -> 用户态
    - 执行一条特权指令--修改PSW的标志位为“用户态”
      - 【2012】这个动作意味着操作系统将主动让出CPU使用权
  - 用户态 -> 内核态
    - 由“中断或异常”引发，【硬件】自动完成变态过程
      - 课后题
      - 触发中断信号意味着操作系统将强行夺回CPU的使用权
      - 用户态下执行【陷入指令】会引发一个内部中断
        - 陷入指令是非特权指令
      - 除了非法使用特权指令之外，还有许多事件会触发中断信号。一个共性是，但凡需要操作系统介入的地方，都会触发中断信号
    - 从内存中取数据可能会导致从用户态 → 内核态
    - 计算机通过硬件中断机制完成由用户态到核心态的转换
    - 注意
      - 有用户态进入核心态，不仅状态需要切换，而且所用的堆栈也可能需要由用户堆栈切换为系统堆栈，但这个系统堆栈也是属于该进程的
  - 一个故事
    - ① 刚开机时，CPU为“内核态”，操作系统内核程序先上CPU运行
    - ② 开机完成后，用户可以启动某个应用程序
    - ③ 操作系统内核程序在合适的时候主动让出CPU，让该程序上CPU运行（操作系统内核在让出CPU之前，会用一条特权指令把PSW的标志位设置为“用户态”）
    - ④ 应用程序运行在“用户态”
    - ⑤ 此时，一位猥琐黑客在应用程序中植入一条【特权指令】，企图破坏系统...
    - ⑥ CPU发现接下来要执行的这条指令是特权指令，但自己又处于“用户态”
    - ⑦ 这个非法事件会引发一个中断信号（CPU检测到中断信号后，会立即变为“核心态”，并停止运行当前的应用程序，转而运行处理中断信号的内核程序）
    - ⑧ “中断”使操作系统再次夺回CPU控制权
    - ⑨ 操作系统会对引发中断的事件进行处理，处理完了再把CPU使用权给别的应用程序

中断和异常的概念

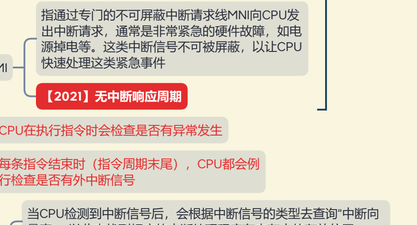
- 中断的作用
  - “中断”是让操作系统内核夺回CPU使用的唯一途径
  - 使CPU从用户态 → 内核态
- 中断（广义）
  - 内中断（异常、例外）
    - 1. 陷阱、陷入trap
      - 例
        - **系统调用指令内含自陷指令**
        - 用户态下执行陷入指令
          - 应用程序想请求操作系统内核的服务，此时会执行一条特殊的指令——【陷入指令】，该指令会引发一个内部中断信号
          - 执行“陷入指令”，意味着应用程序主动地将CPU控制权还给操作系统内核
      - 注意
        - CPU在执行完自陷指令后，自动根据不同的“陷阱”类型进行相应的处理，然后
          - **【2021】返回到自陷指令的【下一条指令执行】**
        - 当自陷指令是转移指令时，并不是返回到下一条指令执行，而是返回到转移目标指令执行
        - 【2020】自陷可用于实现程序调试时的断点设置和单步跟踪
    - 2. 故障
      - 由错误条件引起的，可能被内核程序修复
        - 若可被修复
          - 内核程序修复故障后会把CPU使用权还给应用程序，让它继续执行下去
        - 若不可修复
          - **不能返回到原断点执行，必须终止进程的执行**
            - 【2015】不可继续执行之前异常的指令
        - 例
          - 缺页故障
            - 可回到断点处执行，断点为当前发生故障的指令
          - 非法操作码
            - 试图在用户态下执行特权指令
          - 除数为0
          - 地址越界
          - **存储保护错**
            - 【2016】普通用户访问非法地址空间（OS内核空间）
          - 无法通过异常处理程序恢复故障因此不能回到原断点执行，必须终止进程执行
    - 软件中断（程序性中断）
    - 3. 终止
      - 如果在执行指令的过程中发生了使计算机无法继续执行的【硬件故障】
        - 由致命错误引起，内核程序无法修复该错误，因此一般不再将CPU使用权还给引发终止的应用程序，而是直接终止该应用程序
        - 程序将无法继续执行，只能终止，此时，调出中断服务程序来重启系统
      - **机器校验中断**
        - 如：控制器出错、存储器校验出错
      - 计组：不是由特定指令产生的，而是随机发生的
        - 有矛盾
      - 硬件中断
    - 4. 内中断
      - 异常的检测由CPU自身完成，不必通过外部的某个信号通知CPU
        - **【2015】检测由CPU内部逻辑实现**
      - 若能被修复，则中断处理后返回执行第【i】条指令
        - **产生中断的指令**
  - 外中断（狭义的“中断”）
    - 外中断
      - CPU必须通过中断请求线获取中断的信息，才知道哪个设备发生何中中断
      - 处理完后执行第【i+1】条指令
        - **产生终端的下条指令**
        - 【2021】DMA传送结束
        - 【2021】打印机缺纸
    - 例子
      - 时钟中断
        - 由时钟部件发出的中断信号
      - **I/O中断**
        - 由输入/输出设备发出的中断信号
    - 分为
      - 1. 可屏蔽中断INTR
        - 指通过可屏蔽中断请求线INTR向CPU发出的中断请求。CPU可以通过在中断控制器中设置相应的屏蔽字来屏蔽它或不屏蔽它，被屏蔽的中断请求将不被送到CPU
      - 2. 不可屏蔽中断NMI
        - 指通过专门的不可屏蔽中断请求线MNI向CPU发出中断请求，通常是非常紧急的硬件故障，如电源掉电等。这类中断信号不可被屏蔽，以让CPU快速处理这类紧急事件
        - **【2021】无中断响应周期**
    - 与当前执行的指令无关，中断信号来源于CPU外部
  - 中断机制的基本实现原理
    - 检查中断信号
      - 内中断
        - CPU在执行指令时会检查是否有异常发生
      - 外中断
        - 每条指令结束时（指令周期末尾），CPU都会例行检查是否有外中断信号
    - 计组：找到相应的中断处理程序
      - 当CPU检测到中断信号后，会根据中断信号的类型去查询“中断向量表”，以此来找到相应的中断处理程序在内存中的存放位置
      - 不同的中断信号，需要用不同的中断处理程序来处理
        - **中断处理程序一定是内核程序，需要运行在“内核态”**
      - 中断服务程序的最后一条指令通常是一条中断返回指令，返回到原程序的断点处，以便继续执行原程序
        - **中断返回指令是特权指令**
  - 大多数教材、试卷中，“中断”特指狭义的中断，即外中断
    - 而内中断一般称为“异常”

系统调用

- 概念
  - 系统调用是操作系统提供给应用程序（程序员/编程人员）使用的接口，可以理解为一种可供应用程序调用的特殊函数，应用程序可通过系统调用来请求获得操作系统内核的服务
    - **程序接口由一组系统调用组成**
  - 应用程序通过系统调用请求操作系统服务。而系统中的各种共享资源都由操作系统内核统一管理，因此，凡是与资源共享有关的操作（如存储分配、I/O操作、文件管理等），都必须通过系统调用的方式向操作系统内核提出服务请求，由操作系统内核代为完成。
    - 这样可以保证系统的稳定性和安全性，防止用户进行非法操作
- 系统调用与库函数的区别
  - 操作系统与应用程序之间再细分

  - **不涉及系统调用的库函数：**
    - 取绝对值
    - random
  - 涉及系统调用的库函数：
    - 如“创建一个新文件”的函数folk
    - fork创建进程

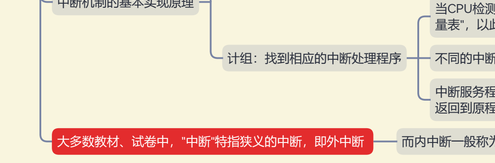
普通应用程序	可直接进行系统调用，也可使用库函数。有的库函数涉及系统调用，有的不涉及
编程语言	向上提供库函数。有时会将系统调用封装成库函数，以隐藏系统调用的一些细节，使程序员编程更加方便。
操作系统	向上提供系统调用，使得上层程序能请求内核的服务
裸机	



- 系统调用（按功能分类）

设备管理	完成设备的 请求/释放/启动 等功能
文件管理	完成文件的 读/写/创建/删除 等功能
进程控制	完成进程的 创建/撤销/阻塞/唤醒 等功能
进程通信	完成进程之间的 消息传递/信号传递 等功能
内存管理	完成内存的 分配/回收 等功能

- **系统调用的过程**
  - **传递系统调用参数 → 执行陷入指令（用户态）→ 执行相应的请求内核程序处理系统调用（核心态）→ 返回应用程序**
  - 注意
    - **1. 陷入指令是在用户态执行的，执行陷入指令之后立即引发一个内中断，使CPU进入核心态**
      - 应用程序通过陷入指令引发中断，转入相应的中断处理程序【即系统调用的入口程序】，请求操作系统服务
        - 处理陷入指令引发的中断处理程序也叫【处理系统调用的内核程序】
      - 陷入指令作为中介
    - **2. 【发出】系统调用请求是在用户态，而对系统调用的相应【处理】在核心态下进行**



- 系统调用指令是对一系列指令序列的封装
- 看到一条高级语言要想到汇编代码
- 【2023】系统调用完成后返回用户态

Presented with xmind