《操作系统》

课程讲稿

第4章 第1讲

中断与异常基础知识

软件所制

第四章 第1讲 中断与异常基础知识

**学时：1**学时

**教学目的：**学习中断与异常机制中的一些基本概念，对整个机制有个大概的了解……

**课程时间线：**

章节简介

中断与异常概念

中断识别

中断服务

5min

15min

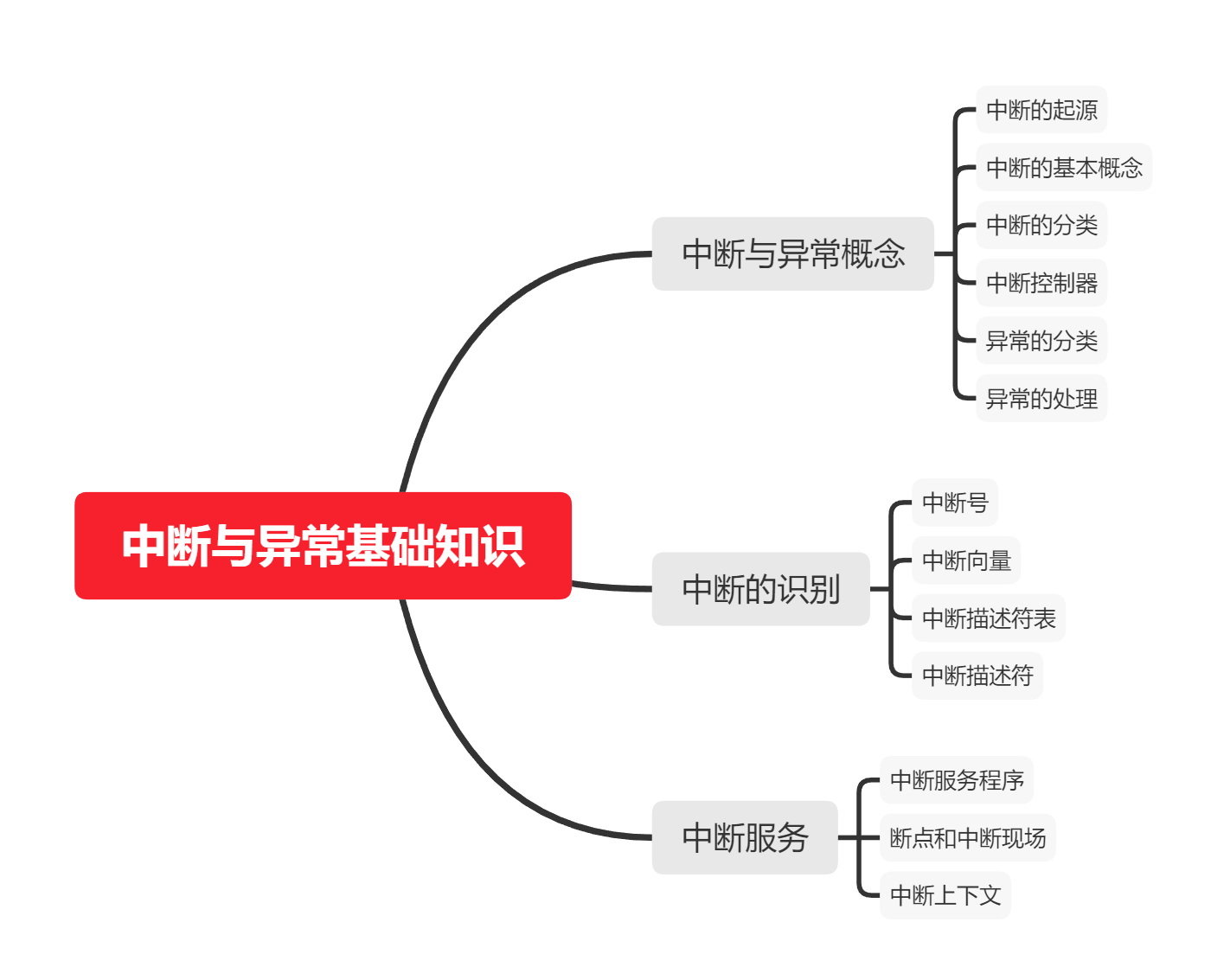
30min

45 min

**课外参考读物：**

《深入理解LINUX内核》

**知识框图：**



**PPT讲稿：**

1. 在第四章中我们将介绍关于操作系统，中断与异常相关的知识。
2. 这是我们这一章的基本内容，分为6节。在本节课中我们将介绍第一节，中断与异常基础知识。
3. 本节中我们将分为三块内容来介绍中断与异常基础知识，分别是中断与异常概念、中断的识别和中断服务，接下来我们先看中断与异常概念
4. 计算机要完成人给它交代的任务，就必须要与人进行交互，或者操作除了cpu、内存之外的设备。例如人操作的设备有：鼠标、键盘；而其他的设备有：硬盘、打印机、显示屏等。而为了与计算机正确的通信，这些设备就必须具备向cpu发送请求的功能，例如键盘向cpu报告：我发送了A、B两个字符。而这些设备要怎么与计算机进行通信呢？换句话说，计算机需要如何知道这些设备发送了请求呢？这在早期的计算机上是使用轮询机制来解决的，所谓轮询，也就是cpu按顺序一遍又一遍询问各个设备，看这些设备是否有请求，如果有则处理，没有则继续当前的任务。可想而知，这样当然会导致以上的两个问题：
   1. CPU一遍又一遍查询可能会发生中断的设备。
   2. 占用太多CPU时间，资源浪费
5. 那么如何解决这些问题呢？我们可以从以下两个方向去思考这个问题：
   1. cpu能不能高冷一点？不去主动询问外设。
   2. 外设能不能主动一点？主动向CPU报告。

我们可以总结一下解决这个问题的思路：首先，cpu只管处理自己当前的任务，但是cpu还是要留一只耳朵来注意外设是否有请求。同样，外设也不用等cpu来询问，当有请求的需要时就直接告诉cpu。而cpu听到有外设发送请求时，则放下当前的任务，转去处理外设的请求，当完成外设的请求后再回到被放下的任务继续执行。

这样，就把之前轮询的任务从cpu分散给各外设了，既降低了cpu的压力，也提高了cpu的利用率，一举两得。

1. 那么，对于前面提出的解决思路，如何实现呢？

既然要搁置当前任务，先处理紧急任务，就需要一个机制来保证当前任务和紧急任务的正常运行。

1. 对于前述的问题，我们使用中断与异常机制来解决问题：

当一个中断信号达到时，CPU必须停止当前正在做的事情，并且切换到一个新的活动。在内核态堆栈保存程序计数器的当前值（即eip和cs寄存器的内容），并把与中断类型相关的一个地址放进程序计数器等待，关于这些寄存器的知识我们留到下一章讲解。

在我们引入中断机制后就可以实现如下的几个功能了：

并行操作：CPU与外设并行进行

实时处理：在限制时间内响应实时进程

故障处理：对于程序运行中出现的故障及时处理

1. 下面我们来看看中断的分类，首先根据中断来自计算机内部还是外部可以分为异常或者中断；第二，在中断中还可以继续细分，根据是否可屏蔽，可分为可屏蔽中断和非屏蔽中断；最后根据发生的时机可以分为同步中断或者异步中断。
2. 在前面在前前面我们所讲的终中和异常，也其实就是异步中断和同步中断。断谓同不中中断，就是在指令执行的某个固定时期才有可能发生的中断，而异步中断就是在指令执行的各个时期都有可能发生的中断。
3. 下面我们来看看，可屏蔽中断和非屏蔽中断的不同。首先内首先在计算机内部也就里面，我们保存了一个屏蔽字。这个屏蔽字的每一个位就代表一个中断。关于这个屏蔽字，我们将在之后的章节进行讲解。除了受本身的屏蔽位的控制之外所有可屏蔽中断都还要受一个总的控制，这个总的控制是由x86CPU的中断允许标志为if。若If为一的话，则CPU可以响应，外屏部的可屏蔽中断。否则不可以响应任何可屏蔽中断。
4. 中断控制器用于控制外中断（可屏蔽中断）请求的发送。

可屏蔽中断向CPU发出请求必须经过中断控制器，而非屏蔽中断以及异常向cpu发送中断请求不用通过中断控制器，也就不受中断控制器的控制了。

1. 中断控制都是用8259a这个中断控制器来对外部中断，也就是可屏蔽中断进行控制。每个8259a中段控器都可以控制8个外部中断通，通常是两片8259a进行级联，这样他们就可以共同控制15个外部中断。
2. 中断控制器 8259A 循环执行如下操作。

（1）监视中断线，检查产生的中断请求（IRQ）信号。

（2）如果在中断线上产生了一个中断请求信号。

a. 把接受到的 IRQ 信号转换成一个对应的向量。

b. 把这个向量存放在中断控制器的一个 I/O 端口，从而允许CPU 通过数据总线读此向 量

c. 把产生的信号发送到 CPU 的 INTR 引脚——即发出一个中断。

d. 等待，直到 CPU 确认这个中断信号，然后把它写进可编程中断控制器（PIC）的一 个 I/O 端口。清 INTR 线。

（3）返回到第一步。

1. 异常是CPU内部产生的中断，即在CPU执行特定指令的时候出现的非法情况（如除数为0等），所以不可能在执行指令期间发生异常，只会在执行一条指令后有可能发生，所以也称同步中断。主要分为以下三种：陷阱、错误和中止。其中陷阱是可以用来完成某些功能的，而错误顾名思义就是运行过程中发生了不可预料的软件类错误，中止就是硬件类错误。
2. 下面我们来了解各种异常之间的区别。其中的EIP，就是指令指针寄存器，存放的是指令的位置。由它们之间的区别，我们可以看出：trap处理完成之后会回到下一条指令运行；而fault处理完成之后，会回到产生异常的本条指令运行。
3. 计算机在运行过程中产生了异常，我们不可能就这样置之不理，肯定要有对应的方法来处理这些异常才能使计算机免于崩溃的危险。因此操作系统必须为每种异常提供一个专门的异常处理程序。
4. 如上表所示，每种异常都有对应的处理程序。在表的最后一列是信号，在这里我们先不讲信号的意义，在后面我们将专门拿一节课来讲解信号的机制。这里我们只需要将信号理解为对进程的一种通知，通知进程接下来应该做什么。
5. 下面我们来看看操作系统是如何识别不同的中断的。
6. 当有中断请求到来的时候，操作系统如何识别是何种中断呢？Linux中为每种中断绑定了一个中断号，就像每种中断的ID号，这就使操作系统能识别每种中断了。
7. 中断向量其实就是中断号，用于选择中断服务例程。操作系统在接收到情断请求后，就会根据这个中断号在内核的一套表结构中寻找这个中断号对应的中断服务例程，来处理这个中断请求。
8. 那么为这些中断分配对应的中断号的时候有没有什么讲究呢？其实在Linux内核程序员在对每个不同的架构进行兼容的时候就已经解决这个问题了，当然在每个架构下有不同的分配策略，但是为了屏蔽底层架构或者硬件的不同，Linux直接对所支持的256个中断向量进行了如下的规定：从 0~31 的向量对应于异常和非屏蔽中断；从 32~47 的向量（即由 I/O 设备引起的中断）分配给屏蔽中断；剩余的 48~255 的向量用来标识软中断。
9. 在前面我们了解到支持系统会根据中断号在内核的一套表结构中寻找这个中断号对应的中断服务例程，而这套表结构有一个专业的名字：中断描述符表。中断号其实就是其对应中断向量所在中断向量表中的位置。
10. 在后面我们可以知道，每个中断描述符占据8字节的位置，而IDT中要存放256个中断描述符，所以IDT要占据2k的位置。
11. 我们知道了在保护模式下的中断描述符也叫做门描述符，在x86下的门描述符总共有3种：任务门，中断门和陷阱门。它们之间的区别就如ppt上所讲。
12. 在Linux下又对这些门描述符进行了细分。事实上，Linux下的中断门和系统中断门就是使用的x86 下的中断门来完成的，系统们和陷阱门也是对x86下的陷阱门进行的细分。
13. 中断到来之后肯定不能让他放着而不管他，肯定要由对应的方法来处理中断对应的请求，因此我们最后再来看看中断服务。
14. 中断描述符表、中断号和中断描述符的作用就是中断请求与中断服务例程之间联系的桥梁。在响应中断的过程种，中断服务例程才是主要干活的那一位。
15. 在整个响应中断的过程中，要涉及到如何使被中断程序的运行不出错的问题。这就需要了解断点、中断现场以及中断上下文的知识了。断点其实就是CPU执行的当前程序被中断时的下一条指令地址，为什么要保存这个断点呢？
16. 为了使被中断程序的运行不出错，就要保存它的运行状态，这个运行状态反映到物理上就是当前CPU内部各寄存器的值。中断现场其实就是当前cpu各寄存器的值，那么为何各寄存器的值就能反映中断现场呢？
17. 中断上下文就是Linux为了运行中断服务例程而在内核中开辟的一个新的空间，这里可以思考一下，为社么要在内核当中开辟这个空间而不在用户空间开辟呢？这是因为中断服务例程对内核具有较大的修改权限，而如果放在用户空间运行的话就可能会面临较大的安全性问题，对整个内核造成威胁。