## 中断：

1. 概念：中断通常被定义为一个事件，该事件改变处理器执行的指令顺序。这样的事件与CPU芯片内外部硬件产生的电信号相对应。
2. 中断的分类：
3. 同步中断：同步中断是当指令执行时由CPU控制单元产生的，之所以称为同步，是因为在一条指令终止执行后CPU才会发出中断。又称为异常。
4. 异步中断：异步中断是由其他硬件设备按照CPU时钟信号随机产生的。又称为中断。
5. 中断信号：
6. 概念：中断信号提供了一种特殊的方式，使处理器转而去运行正常控制流之外的代码。当一个中断信号到达的时候，CPU必须停止它当前正在做的事情，并且切换到一个新的活动。为了做到这一点，就要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），并把中断类型相关的一个地址放进程序计数器。
7. 由中断或异常处理程序执行的代码不是一个进程。更确切的说，它是一个内核控制路径，代表中断发生时正在运行进程执行。作为一个内核控制路径，中断处理程序比一个进程要“轻”（ 中断处理的上下文很少，建立或终止中断处理需要的时间很少）。
8. 中断处理，需要满足的约束：
9. 内核响应中断后需要进行的操作分为两部分：关键而紧急的部分，内核立即执行；其余推迟的部分，内核随后执行。
10. 因为中断随时会到来，所以内核可能在处理其中一个终端时，另外一个中断（不同类型）有发生了。应该尽可能多地允许这种事情发生，因为这能维持更多的I/O设备处于忙状态。因此，中断处理程序必须编写成使相应的内核控制路径能以嵌套的方式执行。当最后一个内核控制路径终止时，内核必须能恢复被中断进程的执行，或者，如果中断信号已导致重新调度，内核能切换到另外的进程。
11. 内核代码中存在一些临界区，在临界区中，中断必须禁止。
12. 中断的分类：
13. 可屏蔽中断：I/O设备发出的所有中断请求，都会产生可屏蔽中断。可屏蔽中断可以处于两种状态：屏蔽的或非屏蔽的；一个屏蔽的中断只要还是屏蔽的，控制单元就忽略它。
14. 非屏蔽中断：只有几个危急事件才引起非屏蔽中断。非屏蔽中断总是由CPU辨认。
15. 异常的分类：
16. 处理器异常：当CPU执行指令时，探测到一个反常条件所产生的异常。可以进一步分为三组，这取决于CPU控制单元产生异常时保存在内核态堆栈eip寄存器中的值。
17. 故障：通常可以纠正，一旦纠正，程序就可以在不失连贯性的情况下重新开始。保存在eip中的值是引起故障的指令地址，因此，当异常处理程序终止时，那条指令会被重新执行。只要处理程序能纠正引起异常的反常条件，重新执行同一条指令就是必要的。
18. 陷阱：在陷阱指令后立即报告；内核把控制权返回给程序后就可以继续它的执行而不失连贯性。保存在eip中的值是一个随后要执行的指令地址。只有当没有必要重新执行已终止的指令时，才出发陷阱。陷阱的主要用途是为了调试程序。在这种情况下，中端信号的作用是通知调试程序一条特殊的指令已被执行（例如到了一个程序内的断点）。一旦用户坚持到调试程序所提供的数据，她就可能要求被调试程序从下一条指令重新开始执行。
19. 异常中止：发生了一个严重的错误；控制单元出了问题，不能再eip寄存器中保存引起异常的指令所在的确切位置。异常中止用于报告严重的错误，如硬件故障或系统表中无效的值。由控制单元发送的这个中断信号是紧急信号，用来把控制权切换到相应的异常中止处理程序，这个异常中止处理程序除了强制受影响的进程终止外，没有别的选择。
20. 编程异常：在编程者发出请求时发生。是由int或int3指令处罚的；当into（检查溢出）和bound（检查地址出界）指令检查的条件不为真时，也引起编程异常。控制单元把编程异常作为陷阱来处理。编程异常通常也叫做软中断。这样的异常有两种常用的用途：执行系统调试用及给调试程序通报一个特定的事件。
21. 每一个中断和异常是由0~255之间的一个数来标识。因特尔把这个8位的无符号整数叫做一个向量。
22. 引脚：由集成电路（芯片）内部电路引出与外围电路的接线。
23. IRQ和中断：
24. 每个能够发出中断请求的硬件设备控制器都有一条名为IRQ的输出线。所有现有的IRQ线都与一个名为可编程中断控制器（PIC）的硬件电路的输入引脚相连
25. 高级可编程中断控制器（APIC）：P152
26. 每一个本地APIC都有一个可编程任务优先级寄存器TPR，TPR用来计算当前运行进程的优先级。
27. 多APIC系统还允许CPU产生处理器间中断。当一个CPU希望把中断发给另一个CPU时，它就在自己本地APC的中断指令寄存器（ICP）中存放这个中断向量和目标本地APIC的标识符。然后，通过APIC总线向目标本地APIC发送一条消息，从而向自己的CPU发出一个相应的中断。
28. 处理器间中断（IPI）由Linux有效地用来在CPU之间交换信息。

## 异常：

1. 异常是由程序错误产生的，或者是由内核必须处理的异常条件产生的。内核为每种异常提供一种专门的异常处理程序。对于某些异常，CPU控制单元在开始执行异常处理程序前会产生一个硬件出错码，并且压入内核态堆栈。

## 中断描述符表（IDT）：

1. 概念：中断描述符（IDT）是一个系统表，它与每一个中断或异常的向量相联系，每一个向量在表中有相应的中断或异常处理程序的入口地址。内核在允许中断发生前，必须适当的初始化IDT。
2. Idtr寄存器使IDT可以位于内存的任何地方，它指定IDT的线性基地址及其最大长度。在允许中断之前，必须用lidt汇编指令初始化idtr。
3. IDT包含3中类型的中断描述符：
4. 任务门描述符：当中断信号发生时，必须取代当前进程的那个进程的TSS选择符存放在任务门中。
5. 中断门描述符：包含段选择符和中断或异常处理程序的段内偏移量。当控制权转移到一个适当的段时，处理器清IF标志，从而关闭将来会发生的可屏蔽中断。
6. 陷阱门描述符：与中断门相似，只是控制权传递到一个适当的段时处理器不修改IF标志。

// Linux利用中断门处理中断，利用陷阱门处理异常。

## 中断和异常的硬件处理：

1. CPU控制单元如何处理中断或异常：
2. 确定与中断或异常关联的向量i
3. 读idtr寄存器指向的IDT表中的第i项。
4. 从gdtr寄存器获得GDT的基地址，并在GDT中查找，以读取IDT表项中的段选择符所标识的段描述符。这个描述符指定中断或异常处理程序所在段的基地址。
5. 确信中断是由授权的中断发生源发出的。
6. 检查是否发生了特权级的变化
7. 如果故障已经发生，用引起异常的指令地址装载入cs和eip寄存器，从而使得这条指令能再次被执行。
8. 在栈中保存eflags、cs及eip的内容
9. 如果异常产生了一个硬件出错码，则将它保存在栈中。
10. 装载cs和eip寄存器，其值分别是IDT表中第i项门描述符的段选择符和偏移量字段。这些值给出了中断或者异常处理程序的第一条指令的逻辑地址。

// 处理完中断信号后，控制单元所执行的指令就是被选中的处理程序的第一条指令。

## 中断和异常处理程序的嵌套执行：

1. 内核控制路径可以任意嵌套：一个中断处理程序可以被另一个中断处理程序“中断”，因此引起内核控制路径的嵌套执行。如果嵌套深度大于1，这些指令将执行上次被打断的内核控制路径，此时的CPU依然运行在内核态。
2. 允许内核控制路径嵌套执行必须付出代价，那就是中断处理程序必须永不阻塞。也就是说，中断处理程序运行期间不能发生进程切换。事实上，嵌套的内核控制路径恢复执行时所需要的所有数据都存放在内核堆栈中，这个栈毫无疑问的属于当前进程。
3. 缺页异常：发生在内核态。这发生在进程试图对属于其地址空间的页进行寻址，然而该页现在不在RAM中时。当处理这样的一个异常时，内核可以挂起当前进程，并用另外一个进程代替它，直到请求的页可以使用为止。只要被挂起的进程又获得处理器，处理缺页异常的内核控制路径就恢复执行。
4. 一个中断处理程序可以抢占其他的中断处理程序，也可以抢占异常处理程序。相反，异常处理程序从不抢占中断处理程序。在内核态能触发的唯一异常就是刚刚描述的缺页异常。
5. 在多处理器系统上，几个内核控制路径可以并发执行。此外，与异常相关的内核控制路径也可以开始在一个CPU上执行，并且由于进程切换而转移到另外一个CPU上执行。

## 初始化中断描述符表：

1. 内核启用中断以前，必须把IDT表的初始地址装到idt寄存器，并初始化表中的每一个项。这项工作是在初始化系统时完成的。
2. Linux系统的中断描述符：
3. 中断门：用户态的进程不能访问的一个Intel中断门（门的DPL字段为0）。所有的Linux中断处理程序都通过中断门激活，并全部限制在内核态。
4. 系统门：用户态的进程可以访问的一个Intel陷阱门（门的DPL字段为3）。通过系统门来激活三个Linux异常处理程序，它们的向量是4，5，128。
5. 系统中断门：能被用户进程访问的Intel中断门（门的DPL字段为3）。与向量3相关的异常处理程序由系统中断门激活的。
6. 陷阱门：用户态的进程不能访问的一个Intel陷阱门（门的DPL字段为0）。大部分Linux异常处理程序通过陷阱门来激活。
7. 任务门：不能被用户态进程访问的Intel任务门（门的DPL字段为0）。Linux对“缺页”异常的处理程序是由任务门激活的。
8. IDT的初步初始化：
9. 当计算机运行在实模式时，IDT被初始化并由BIOS例程使用。然后，一旦Linux接管，IDT就被移动到RAM的另外一个区域，并进行第二次初始化，因为Linux没有利用任何BIOS例程。

## 中断处理：

1. 中断类型：
2. I/O中断：某些I/O设备需要关注；相应的中断处理程序必须查询设备以确定适当的操作过程。
3. 时钟中断：某种时钟产生一个中断，这种中断告诉内核一个固定的时间间隔已经过去了。这些中断大部分是作为I/O中断来处理的。
4. 处理器间中断：多处理器系统中一个CPU对另外一个CPU发出一个中断。
5. 中断向量：
6. 向量（IRQ号）与设备之间的对应是在初始化每个设备驱动程序时建立的。
7. 如果一个出现中断，内核没有处理，那么这个中断就是意外中断。也就是输，与某个IRQ线相关的中断处理例程（ISR）不存在。

## IRQ在多处理器系统上的分发：

1. 当硬件设备产生了一个中断信号时，多APIC系统就选择其中的一个CPU，并把该信号传递给相应的本地APIC，本地 APIC又中断它的CPU、这个事件不通报给其他所有的CPU。
2. I/O APIC：
3. 一组24条IRQ线，一张24项的中断重定向表，可编程寄存器，通过APIC总线发送和接收APIC信息的一个信息单元。中断重定向表中的每一项都有可以被单独编程以指明中断向量和优先级、目标处理器以及选择处理器的方式。重定向表中的信息用于把每个外部IRQ信号转换为一条信息，然后，通过APIC总线把消息发送给一个或者多个本地APIC单元。
4. 作用：接收外部的中断事件，然后传给LOCAL APIC
5. 来自外部硬件设备的中断请求以两种方式在可用CPU之间分发：
6. 静态分发：IRQ信号传递给重定向表相应项中所列出的本地APIC。中断立即传递给一个特定的CPU，或者一组CPU，或者所有的CPU。
7. 动态分发：如果处理器正在执行最低优先级的进程，IRQ信号就传递给这种处理器的本地APIC。
8. LOCAL APIC的作用：
9. 从CPU的int、pin接收中断，或者I/O APIC会发送中断信息给LOCAL APIC。
10. 在多处理器中间，它会发送或接收处理器之间的中断信息。
11. 中断服务例程：一个中断服务例程（ISR）实现一种特定设备的操作。
12. 开中断：进程的执行可以被中断。
13. 关中断：是为了保护一些不能中途停止执行的进程而设计的。

## 软中断及tasklet：

1. 软中断：
2. 常常表示可延迟函数的所有种类。
3. 软中断的分配是静态的（即在编译时定义），而tasklet的分配和初始化可以在运行时进行。软中断可以并发地运行在多个CPU上。因此，软中断是可重入函数（可以被中断的函数）并且必须明确地使用自旋锁保护其数据结构。Tasklet不用担心这些问题，因为内核对tasklet的执行进行了更加严格的控制。相同类型的tasklet总是被串行地执行，换句话说：不能在两个CPU上同时运行相同类型的tasklet。但是，类型不同的tasklet可以在几个CPU上并发执行。Tasklet串行化使得tasklet函数不必是可重入的。
4. 中断上下文：表示内核当前正在执行一个中断处理程序或一个可延迟的函数。
5. tasklet：是Linux内核中可延迟机制中的一种。（可延迟函数）。软中断和tasklet有密切的关系，tasklet是在软中断之上实现的。

## 工作队列：

1、工作队列是把工作堆后执行的一种机制。

2、可延迟函数和工作队列非常相似，区别还是很大的，主要区别：

（1）延迟函数运行在中断上下文中，而工作队列中的函数运行在进程上下文中。执行可阻塞函数（比如需要访问磁盘数据块的函数）的唯一方式是在进程上下文中运行。在中断上下文中不可能发生进程切换。可延迟函数和工作队列中的函数都不能访问进程的用户态地址空间。事实上，可延迟函数被执行时不可能有任何正在运行的进程。另一方面，工作队列中的函数是由内核线程来执行的，因此，根本不存在它要访问的用户态的地址空间。

3、NR\_CPUS是系统中CPU的最大数目

4、工作队列由工作者线程的特殊内核线程来执行。每个工作者线程在worker\_thread()函数内部不断地执行循环操作。

5、预定义工作队列（events）：只是一个包括不同内核层函数和I/O驱动程序的标准工作队列。

## 从中断或异常返回：

1. 中断处理程序结束时，内核进入ret\_from\_intr()；而当异常处理程序结束时，它进入ret\_from\_exception()。