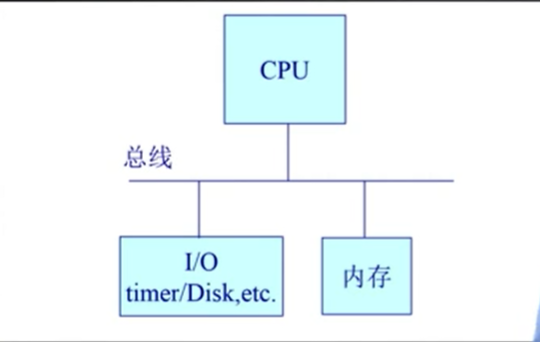


启动：



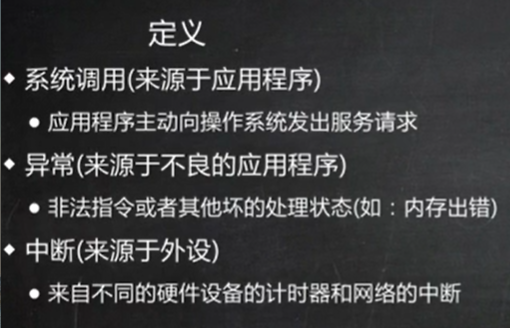
DISK：存放OS

BIOS：基本I/O处理系统

Bootloader：加载OS，将OS从硬盘放到内存处运行

POST（加电自检）寻找显卡和执行BIOS

Bootloader对系统的启动过程没太明白



应用程序必须通过操作系统才能访问外设，原因：

在计算机运行中，内核是被信任的第三方；

只有内核可以执行特权指令；

为了方便应用程序

操作系统如何设计和实现中断、异常和系统调用？三者区别和特点

产生来源：

中断：外设；

异常：应用程序意想不到的行为

系统调用：应用程序请求操作系统提供服务

处理时间：

中断：异步（当一个所谓的异步事件产生时，应用程序并不知道这个事件什么时候产生）

异常：同步（当应用程序执行到某一条指令后一定会产生）

系统调用：异步或同步（就可以理解为是不是同一个时间点）

应用程序发出请求后等待操作系统的系统服务，属于同步；

应用程序发出请求后又立马去做其他事情，则属于异步；

响应：

中断：持续，对用户应用程序是透明的；

异常：杀死或重新执行意想不到的的应用程序指令；

系统调用：等待和持续

中断的一些操作：

硬件上保护：设置中断标记[CPU初始化]，将内部外部事件设置中断标记和中断事件ID

软件上保护（就是操作系统）：保存当前处理状态、中断服务程序处理、清除中断标记、恢复之前保存的处理状态，（整个过程是应用程序透明的）

异常的一些操作：

异常编号：保存现场、异常处理（杀死产生了异常的程序、重新执行异常指令）、恢复现场（重新执行异常指令时对用户应用程序也是透明的）

系统调用

应用程序必须要操作系统的帮助才能完成的服务，需要系统调用，可以理解为接口

程序访问主要是通过高层次的API接口而不是直接进行系统调用

操作系统调用实现：

通常情况下，与每个系统调用相关的序号，系统调用接口根据这些序号来维护表的索引；

系统调用接口调用内核态中预期的系统调用，并返回系统调用的状态和其他任何返回值；

用户不需要知道系统调用是如何实现的，只需要获取API和了解操作新系统将什么作为返回结果，操作系统接口的细节大部分都隐藏在API中，通过运行程序支持的库来管理(用包含编译器的库来创建函数集)

用户态，CPU所处的特权级的状态

内核态，CPU可以处理特权指令

系统调用与函数调用有区别：

当应用程序发出函数调用时是在一个栈空间完成参数传递和返回；

当应用程序发出系统调用时，应用程序和系统内核拥有各自的堆栈，当应用程序发出系统调用切换到内核态去执行时，需要切换切换到相应的堆栈上，同时需要完成特权级的转换：既从用户态转换到内核态，这些切换转换需要一定的系统开销，因此系统调用要比函数调用花费的开销大，回报是安全、可靠的服务；

