**一、操作系统的两个主要功能是什么？**

**答：**

**角度1：从任务角度来看**

**1、通过资源管理，提高系统的使用效率，并且控制执行：**

**资源管理**包括处理器管理、存储管理、设备管理、文件管理、联网与通信管理等等。操作系统负责管理计算机的硬件资源，如处理器（CPU）、内存、硬盘、网络适配器等。操作系统需要有效地管理这些资源，以便多个程序可以共享并协同使用计算机系统。

**控制执行**包括并发控制、作业调度、进程调度等等。操作系统通过控制执行来有效地分配和管理计算机系统的资源**；**通过合理的作业调度和进程调度算法，操作系统可以优化系统的性能**；**操作系统控制执行的功能使得计算机可以同时处理多个任务**；**操作系统通过控制执行来确保系统的稳定性和安全性。

**2、改善人机界面,向用户提供友好的工作环境：**

操作系统提供了用户与计算机硬件之间的接口，使用户能够与计算机进行交互。这包括命令行界面（CLI）和图形用户界面（GUI）。通过这些接口，用户可以向计算机发出指令、运行程序、管理文件和文件夹等。操作系统的用户接口通常是用户与计算机之间的桥梁，使得用户可以轻松地使用计算机系统进行各种操作。

**角度2：（老师PPT）自顶向下和自底向上来看**

**1、自顶向下（扩展机）：**

为应用开发人员提供简洁易用的资源抽象

**2、自底向上（资源管理者）：**

管理种类繁多、纷繁复杂的硬件资源

**二、为什么说操作系统内核是中断驱动的？试举例几种典型的中断发生场景。**

**答：**

**为什么：**

这是因为操作系统内核在处理内部事件和外部事件时主要依靠中断机制。中断是一种计算机体系结构中的基本概念，它是多道并发的基础，它允许外部设备（如硬件设备或其他处理器）通过发送信号来打断CPU的正常执行流程，从而请求处理器的服务。在接收到中断信号后，一般而言CPU就会从用户态翻转到核心态，而操作系统是运行在核心态的软件，此时操作系统就会上来处理中断。所以说操作系统内核是中断驱动的。

**几种中断的典型场景：**

**1.时钟中断：**

时钟中断是最常见的中断类型之一。计算机系统中的时钟定期发出中断信号，用于更新系统时间、调度任务和触发定时操作。操作系统内核会在时钟中断发生时进行处理，例如更新系统时间戳、重新安排任务调度等。

**2. I/O设备中断：**

当外部I/O设备（如键盘、鼠标、硬盘等）完成数据传输或需要处理时，它们会向CPU发送中断请求。操作系统内核会在收到这些中断信号后处理相关的I/O操作，**例如**从键盘读取输入、将数据写入硬盘等。

**3.硬件故障中断：**

硬件故障或异常情况（**例如**内存错误、硬盘故障等）可能会导致中断发生。操作系统内核会在检测到这些中断时采取适当的措施，例如向用户报告错误、尝试修复问题或进行系统重启。

**4.程序性中断（包括异常中断）：**

分为三类：语法错误，逻辑错误，程序运行异常（**比如**除以0，定点溢出）。这些中断需要编译程序报错或者测试程序报错或者借助于信号机制，通过操作系统捕获并且转交给相应应用程序处理。

**5.系统调用（访管中断）：**

系统调用是用户程序主动请求操作系统内核提供服务的一种方式。当用户程序使用访管指令、执行系统调用时，会触发一个中断，使操作系统内核能够在用户态和内核态之间切换，并执行用户程序请求的操作，**例如**文件操作、进程管理等。

**三、操作系统开发者一般会提供库函数来封装系统调用，这样做的好处是什么？应用程序开发人员是否可以绕开库函数直接访问操作系统内核提供的系统服务？如果可以，需要了解哪些额外信息？**

**答：**

**1.提供库函数来封装系统调用的好处：**

**1.简化编程、提供接口：**

库函数提供了高级接口，使得应用程序开发人员无需直接与底层的系统调用进行交互，从而简化了编程过程。这使得开发人员可以更专注于实现应用程序的功能，而不必过多考虑底层系统的细节。

**2.具有跨平台兼容性：**

由于不同操作系统的系统调用接口可能有所不同，提供库函数封装可以隐藏这些差异，使得应用程序具有更好的跨平台兼容性。开发人员可以使用相同的库函数来编写跨平台的应用程序，而不必担心不同操作系统的细节差异。

**3.增加安全性和稳定性：**

库函数可以提供额外的安全性检查和错误处理机制，帮助开发人员避免常见的错误，提高应用程序的稳定性和安全性。例如，库函数可能会对输入参数进行验证，以防止发生缓冲区溢出等安全漏洞。

**2.是否可以绕过：**

**严格意义上不可以绕过：**

一般情况下，用户程序不能直接访问系统内核提供的服务。允许开发人员通过程序直接访问系统资源会引发许多问题，带来安全性和稳定性上的问题，所以本身就设置了相应权限。正常渠道下，用户程序必须调用库函数，通过操作系统发出服务请求，由操作系统处理相关代码。如果硬要绕过，也存在对操作系统库函数实现特定细节未知、难以进行错误处理、破坏封装的问题，而产生不必要的风险。

程序的运行空间分为内核空间和用户空间，其程序各自按不同的特权运行，在逻辑上相互隔离。应用程序不能直接访问内核数据，也无法直接调用内核函数，他们只能在用户空间操纵用户数据，调用用户空间函数。但在很多情况下，应用程序需要获得系统服务，这时就必须利用系统提供给用户的特殊接口--系统调用。否则，就会缺失很多关键信息而无从下手。

**3.想要绕过需要的额外的信息：**

**1. 系统调用接口：**

开发人员需要了解系统调用的接口及其参数，包括系统调用的名称、参数的数量和类型、返回值等。这些信息通常会记录在操作系统的文档或手册中。

**2. 操作系统特定细节，如系统结构与资源分配情况：**

不同操作系统的系统调用接口可能会有所不同，因此开发人员需要了解目标操作系统的特定细节。例如，不同操作系统对文件系统、进程管理等方面的实现可能存在差异。用户程序需要了解系统的结构和资源分配情况，以便直接操作硬件或系统资源。

**3. 系统调用的语义：**

开发人员还需要了解系统调用的具体语义，即它们执行的功能和对系统状态的影响。例如，对文件系统进行读写操作的系统调用会有不同的语义，开发人员需要理解它们的具体行为。

**4. 权限和安全性考虑：**

直接访问操作系统内核提供的服务可能需要特殊的权限或者超级用户权限。开发人员需要了解在何种情况下可以调用这些系统调用，以及是否需要特定的权限。此外，开发人员还需要谨慎处理直接访问系统调用可能带来的安全风险。用户程序需要了解如何进行安全的资源访问，以防止意外或恶意操作。

**5.错误处理和异常情况：**

调用系统调用可能会导致各种错误或异常情况的发生，例如输入参数错误、资源不足等。开发人员需要了解如何正确地处理这些错误或异常情况，以确保程序的稳定性和可靠性。

**6.除此之外：**

用户最好还需要一些系统状态和调度、操作系统版本和兼容性、以及需要操作系统手册等等方面的信息，具体情况具体分析。

**四、试比较宏内核和微内核的区别。**

**答：**

**1.本身定义、内核结构、运行模型不同：**

**宏内核：** 将大部分操作系统功能实现为一个单独的、紧密耦合的内核空间。这包括进程管理、内存管理、文件系统、设备驱动等功能，都直接运行在内核空间内。

**微内核：** 微内核采用一种更加模块化的设计，将核心的操作系统功能分割成不同的服务或模块，每个模块运行在独立的用户空间进程中。通常只包含最基本的功能，如调度器、内存管理、进程间通信等，而其他高级功能如文件系统、网络协议栈等则以用户空间服务的形式存在。

**2. 性能不同：**

**宏内核：** 宏内核由于将所有功能集中在一个内核空间，因此其内部通信和数据传输通常更加高效，性能更好。

**微内核：** 微内核需要通过进程间通信来进行模块间的交互，因此微内核通常会比宏内核在性能上略低。

**3. 可靠性和安全性不同：**

**宏内核：** 宏内核的设计通常更加简单直接，内核空间较大较复杂，因此在某些情况下稳定性和可靠性较低。

**微内核：** 微内核的模块化设计使得系统更加灵活，有助于隔离不同功能模块，从而提高了系统的安全性和可靠性。

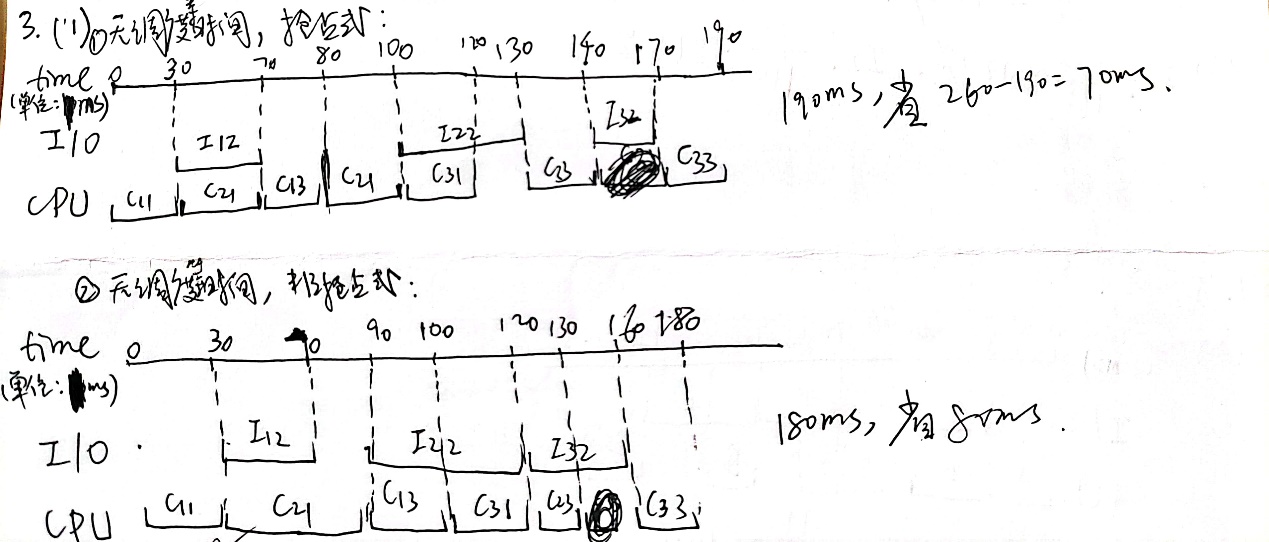
**4. 扩展性和可维护性（灵活性）不同：**

**宏内核：** 宏内核通常较难进行定制和扩展，因为所有功能都集中在一个内核空间中，需要直接修改内核代码才能实现定制功能。

**微内核：** 微内核的模块化设计使得系统更容易进行定制和扩展，可以通过添加、移除或替换不同的模块来实现特定的功能定制，更加灵活。

**5.有三道程序，按照A、B、C的优先次序运行，其内部计算和I/O操作时间如下图所示。试画出多道运行的时间关系图(忽略操作系统执行时间)。完成三道程序共花费多少时间?比单道运行节省多少时间?若操作系统每次执行中断处理、作业调度与切换等任务，每个任务花费1ms，试画出各程序状态转换的时间关系图（分抢占和非抢占两种调度策略）。**

**1.不考虑调度等时间：**



**计算过程：**

**单道运行时间：** 30+40+10+60+30+10+20+40+20=260 ms

**多道抢占式运行时间：**30+40+30+70+20=190 ms 比单道运行**节省**：260-190=70 ms

**多道非抢占式运行时间：**30+60+30+40+20=180 ms 比单道运行**节省**：260-180=80 ms

**2.考虑调度等时间：**

