操作系统

作业一

T1

> 请分别从系统和用户的角度，阐述操作系统的功能，并具体描述操作系统需要提供哪些服务。

系统角度：

操作系统可看做资源分配器，管理 CPU 时间、内存空间、文件存储空间、I/O 设备等资源的使用或分配。

用户角度：

操作系统为用户提供用户界面、程序执行、文件系统操作等服务，让用户更方便的使用计算机资源。

操作系统需要提供的服务：

- 用户界面：可能有多种形式，一是命令行界面，采用文本命令；二是批处理界面，命令或指令可以编成文件以便执行；三是图形界面，是一种视窗系统，可以通过定位设备控制 I/O、通过菜单选择等。
- 程序执行：加载程序到内存，并加以运行。程序应能结束执行，包括正常或不正常。
- I/O 操作：程序运行可能需要 I/O，为了效率和保护，用户通常不应直接控制 I/O 设备，因此操作系统需要提供手段执行 I/O。
- 文件系统操作：需要提供读写文件和目录、创建和删除文件、搜索给定文件等功能。需要权限管理，根据文件所有者允许或拒绝对文件和目录的访问。
- 通信：提供进程间交换信息的渠道，可以通过共享内存或信息交换实现。
- 错误检测：操作系统需要不断检测和更正错误，对于检测出的错误，操作系统必须采取适当动作，确保计算的正确和一致。
- 资源分配：当多个用户或多个作业同时运行时，操作系统管理许多不同类型的资源，负责资源的分配调度。
- 记账：记录用户使用资源的类型和数量，可用于记账或统计使用量。
- 保护与安全：当多个独立进程并发执行时，一个进程不应干预其他进程或操作系统本身，操作系统要确保可以控制系统资源的所有访问。要求用户向系统认证自己，以获取系统资源的访问权限，保护外部 I/O 设备不受非法访问，并记录所有非法的闯入企图。

T2

> 请阐述 multi-programming 和 multi-tasking 的概念与设计目的。

h5 multi-programming

概念：

通过安排作业（编码与数据）使得 CPU 总有一个执行作业。在计算机内存中同时存储多个程序，并让它们在同一时间内轮流执行，当一个程序等待某个资源（比如等待I/O操作）时，CPU可以立即切换到另一个程序，继续执行。

设计目的：

提高 CPU 利用率。

h5 multi-tasking

概念：

是 `multi-programming` 的自然延伸，虽然 CPU 还是通过切换作业来执行多个作业，但由于切换频率很高，用户可以在程序运行时与其交互。

设计目的：

满足用户多任务处理的需求，提高用户体验。

T3

> 请阐述缓存的思想以及工作原理。

思想：

用空间换时间，用更快的存储空间存储经常访问的数据，以便下次快速访问。

工作原理：

当内存中的信息被使用时，它会被临时复制到更快存储系统，即高速缓存；当需要特定信息时，首先检查它是否处于高速缓存，如果是，可以直接使用高速缓存的信息，如果否，就使用位于源地的信息，同时将其复制到高速缓存以便下次再用。

T4

> 请阐述什么是系统调用，以及系统调用与 API 的逻辑关系。

系统调用：

操作系统提供给用户程序的一种接口，通过这种接口，用户程序可以让操作系统执行某些操作，如打开、创建或删除文件等。

API：

即应用编程接口，应用程序开发人员根据 API 来设计程序，API 为方便应用开发人员规定了一组函数，在后台，API 函数为应用程序员调用实际的系统调用。

逻辑关系：

API是一个提供给应用程序的接口，一组函数，是与程序员进行直接交互的，系统调用则不与程序员进行交互的；某些 API 所提供的功能会涉及到与内核空间进行交互。那么，这类 API 内部会封装系统调用。而不涉及与内核进行交互的 API 则不会封装系统调用，一个 API 函数可能包含一个或多个系统调用，也可能不包含系统调用。

T5

> “
> 阐述 Dual Mode 的工作机制，以及采用 Dual Mode 的原因。

工作机制：

计算机系统至少需要两种单独工作模式：用户模式和特权模式。当系统引导时，硬件从特权模式开始，操作系统接着加载，然后在用户模式下执行用户程序。一旦有陷阱或中断，硬件会从用户模式切换到特权模式，在将控制交给用户程序之前，系统会切换到用户模式。模式的切换通过模式位实现，0 和 1 分别代表特权模式和用户模式。

采用原因：

双重模式执行提供了保护手段，将可能引起损害的机器指令作为特权指令，即只有在特权模式下才允许执行的指令，以便防止操作系统和用户程序受到错误用户程序的影响。

T6

> “
> 分别阐述 Monolithic 大内核结构，层次化结构，模块化结构和微内核结构的特点和优劣。

h5 大内核结构：

特点：所有功能都在同一片空间中运行，没有层次和模块化。

优点：性能高，各个模块之间不需要通过消息传递来通信，而可以直接调用函数或共享数据；也不需要特权模式和用户模式的切换。

缺点：可靠性和稳定性低，一个模块的故障可能引起整个系统的崩溃；可扩展性差，没有模块化，操作系统代码的复杂性高，不易维护升级。

h5 层次化结构：

特点：内核分成若干个层次，每个层次都负责不同的功能，每层只能调用更底层的功能和服务。

优点：简化了构造和调试，提高了可维护性，先调试低层再调试高层；每个层次只需关注自己的功能，减少了模块间的耦合性，提升了系统可靠性和稳定性。

缺点：每个层次之间的通信和数据传输增加了系统开销，降低了系统效率；层次化设计需要考虑合理的分层，增加了设计难度。

h5 模块化结构：

特点：将操作系统分为若干个功能模块，每个模块都有确定的功能和接口，模块间通过接口交互。

优点：代码结构清晰，可维护性和可拓展性高；每个模块负责不同功能，耦合性低，易于维护升级。

缺点：模块间的交互也会提升系统开销，降低系统效率；每个模块的功能和接口需要合理设计，增加了设计难度。

h5 微内核结构：

特点：从内核中删除所有不必要的部件，而将它们当做系统级和用户级的程序来实现，微内核之间通过消息传递通信。

优点：可靠性和安全性高，避免外部程序的错误对内核造成影响；内核功能简单，易于维护和管理；可扩展性高，可以方便的在不同平台上运行或新增功能。

缺点：服务进程的系统调用频繁，微内核间消息传递增加开销，性能受损。

T7

> “
> 举例说明什么是机制与策略分离的设计原则，并说明该设计的好处。

设计原则：

机制决定如何做，而策略决定做什么。设计原则是机制与策略分离，即把实现某种功能的基本模型（策略）和如何具体实现（机制）分离开。

例子：

Unix/Linux的接口设计有一句通用的格言“提供机制而不是策略”。在内核中提供基本机制，如进程管理，内存管理，文件系统等，在用户空间中提供多种策略，如shell，编辑器等。

好处：

降低了系统复杂度，机制和策略各自独立，易于设计或维护；增加了系统的灵活性或通用性，策略的改变不一定需要底层机制的改变，同一种机制可能支持多种策略，同一个策略可以适用多种机制。