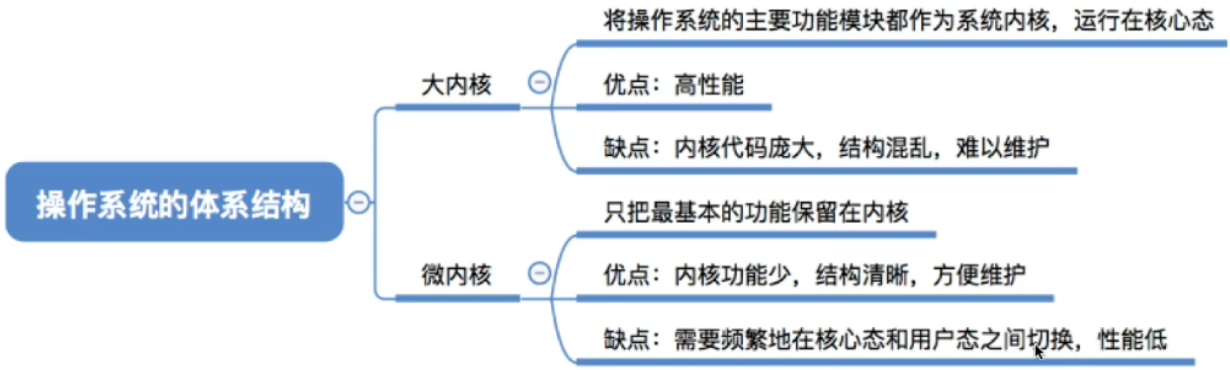
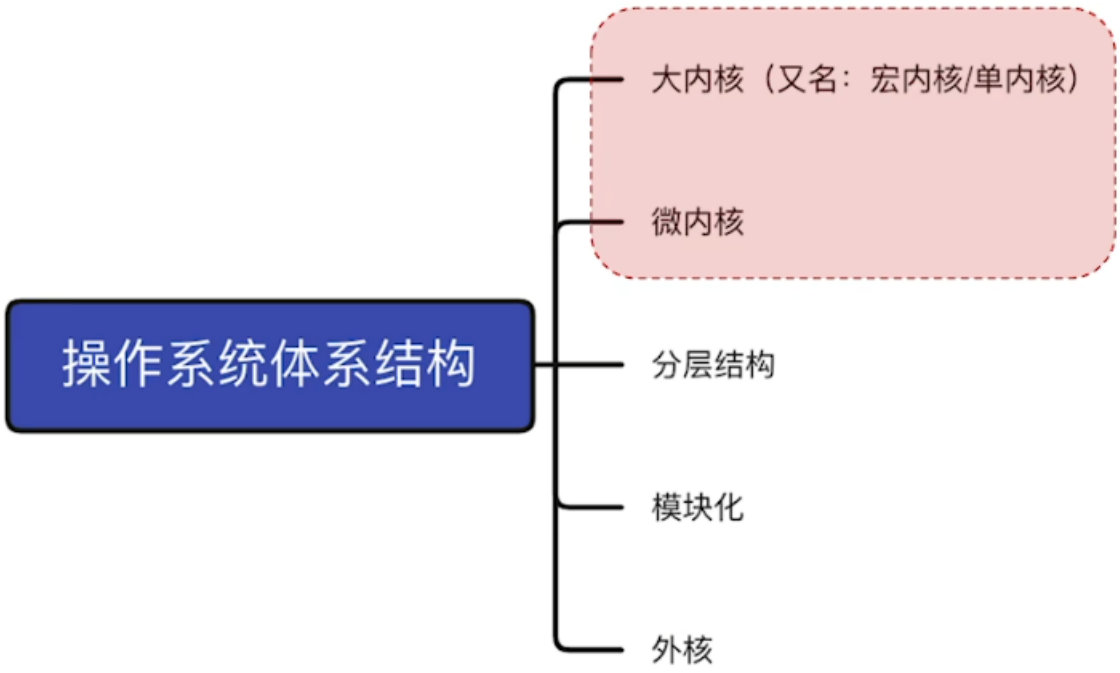
操作系统体系结构

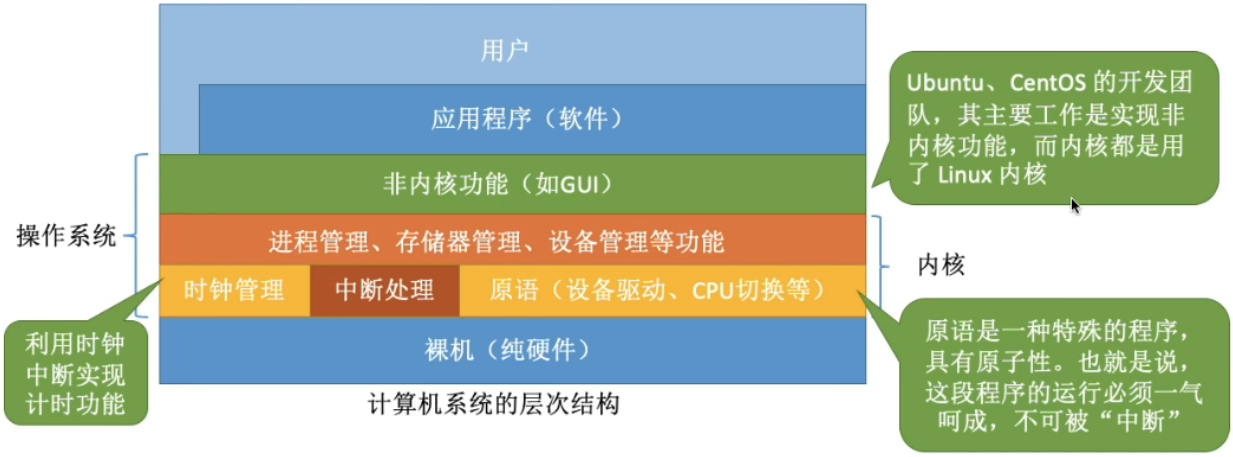


典型的大内核/宏内核/单内核 操作系统： Linux、UNIX
典型的 微内核 操作系统： Windows NT



操作系统的内核

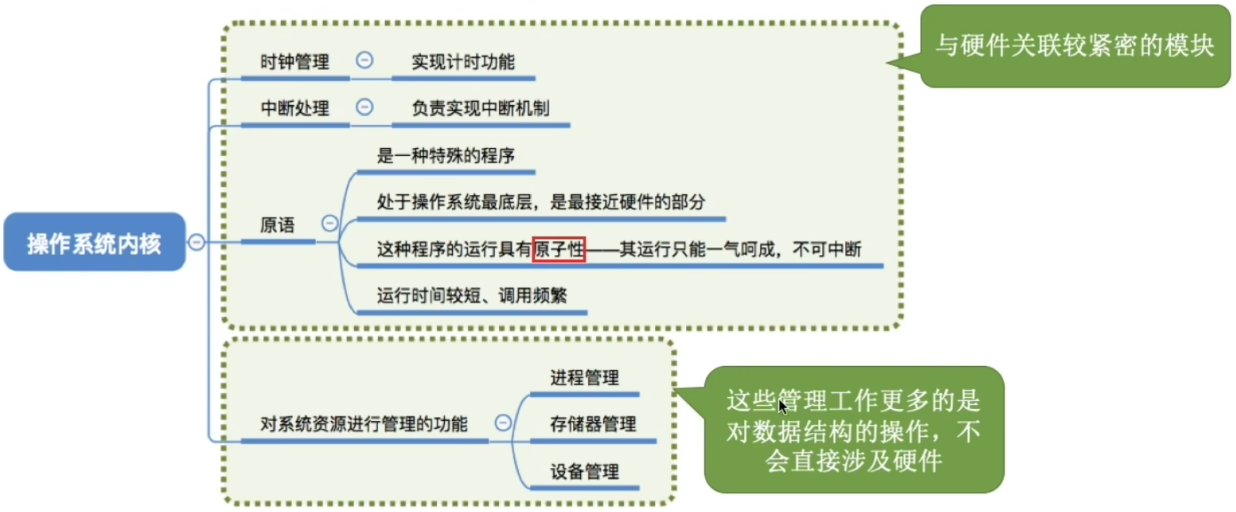
计算机系统的层次结构



计算机系统的层次结构

内核是操作系统最基本、最核心的部分。

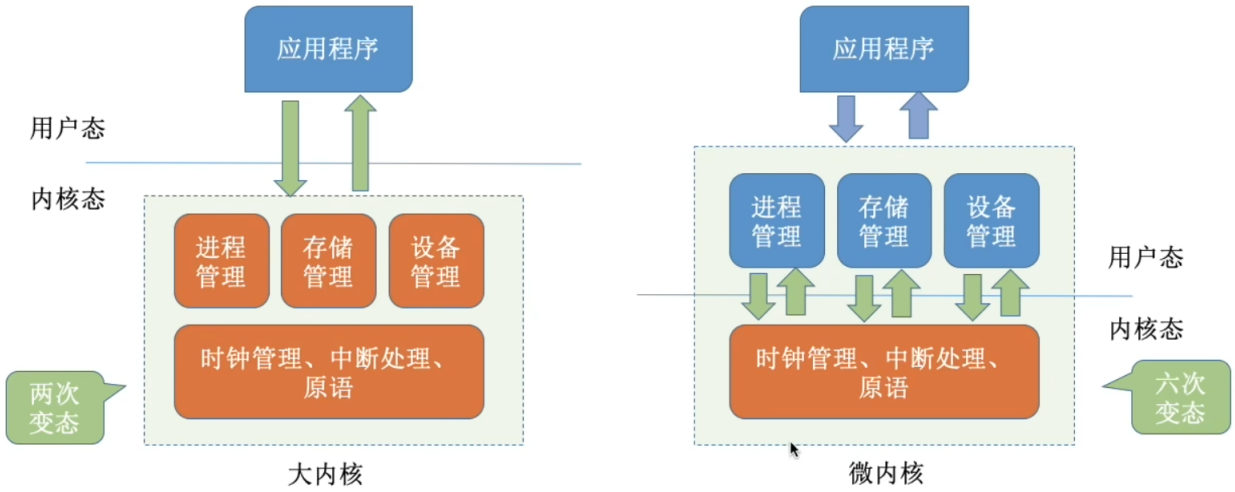
实现操作系统内核功能的那些程序就是内核程序。



计算机系统的层次结构

注意：
操作系统内核需要运行在内核态
操作系统的非内核功能运行在用户态

操作系统的体系结构



一个故事：现在，应用程序想要请求操作系统的服务，这个服务的处理同时涉及到进程管理、存储管理、设备管理
注意：变态的过程是有成本的，要消耗不少时间，频繁地变态会降低系统性能

操作系统结构

	特性、思想	优点	缺点
分层结构	内核分多层，每层可单向调用更低一层提供的接口	1. 便于调试和验证，自底向上逐层调试验证	1. 仅可调用相邻低层，难以合理定义各层的边界
		2. 易扩充和易维护，各层之间调用接口清晰固定	2. 效率低，不可跨层调用，系统调用执行时间长
模块化	将内核划分为多个模块，各模块之间相互协作。 内核 = 主模块+可加载内核模块 主模块：只负责核心功能，如进程调度、内存管理 可加载内核模块：可以动态加载新模块到内核，而无需重新编译整个内核	1. 模块间逻辑清晰易于维护，确定模块间接口后即可多模块同时开发	1. 模块间的接口定义未必合理、实用
		2. 支持动态加载新的内核模块（如：安装设备驱动程序、安装新的文件系统模块到内核），增强OS适应性	2. 模块间相互依赖，更难调试和验证
		3. 任何模块都可以直接调用其他模块，无需采用消息传递进行通信，效率高	
宏内核（大内核）	所有的系统功能都放在内核里（大内核结构的OS通常也采用了"模块化"的设计思想）	1. 性能高，内核内部各种功能都可以直接相互调用	1. 内核庞大功能复杂，难以维护
			2. 大内核中某个功能模块出错，就可能导致整个系统崩溃
微内核	只把中断、原语、进程通信等最核心的功能放入内核。进程管理、文件管理、设备管理等功能以用户进程的形式运行在用户态	1. 内核小功能少、易于维护，内核可靠性高	1. 性能低，需要频繁的切换 用户态/核心态。 用户态下的各功能模块不可以直接相互调用，只能通过内核的"消息传递"来间接通信
		2. 内核外的某个功能模块出错不会导致整个系统崩溃	2. 用户态下的各功能模块不可以直接相互调用，只能通过内核的"消息传递"来间接通信
外核（exokernel）	内核负责进程调度、进程通信等功能，外核负责为用户进程分配未经抽象的硬件资源，且由外核负责保证资源使用安全	1. 外核可直接给用户进程分配"不虚拟、不抽象"的硬件资源，使用户进程可以更灵活的使用硬件资源	1. 降低了系统的一致性
		2. 减少了虚拟硬件资源的"映射层"，提升效率	2. 使系统变得更复杂