多道程序设计: 在内存中同时放几道相互独立的程序，减少IO的开销，CPU使用效率进一步提高，宏观并行，微观串行。不需要人为参与，交互性不够好，衍生出分时系统。

分时系统:使得不同的程序都有时间去执行它的工作，时间片:千分之一秒。

分布式Os:依赖于网络的快速发展，提供给用户前端，经由网络将工作交给DataCenter处理。故Os会产生相应的应对:松、紧耦合系统:松:为了让网络及时有效。DataCenter是紧耦合(集群Os)。

为了使Os更容易扩展，模块与模块之间不采用函数调用，采用类似消息传递的机制(松耦合)，提出微内核架构设计:在OsKernel里尽量放最基本的功能，比如中断，消息传递。像文件系统，内存管理，网络协议栈等等放在外围以进程或程序的方式存在(服务)，服务与服务之间通过kernel的消息传递机制来进行通信。

在kernel通过地址隔离，可以确保相互之间的程序无法恶意破坏对方的地址空间。这种灵活安全的设计方式的代价是性能的开销:比如文件系统和内存管理系统之间进行交互，需要将数据先导入kernel，kernel再导给内存管理的子系统，之间需完成多次拷贝。产业界很少采用微内核架构。

外核:kernel分成两块，Exokernel:完成硬件功能的复制。LibraryOs跟具体应用打交道，应用不同，对应LibraryOs也不一样。LibraryOs访问Exokernel可以安全有效并发的使用硬件资源。

虚拟机监控器(VMM):跑在传统Os之下，虚拟出多个Os。

Bootloader:放在Disk，用来加载Os，能够将Os从Disk放到Memory里，让CPU执行Os。

一般放在主硬盘的第一个引导扇区(512k)。

一开始加电，X86的BIOS在CS:IP = 0xf000:fff0(CS:段寄存器；IP:指令寄存器)地址开始执行，地址执行之后会完成POST(加电自检)，执行完毕之后加载BootLoader，加载到0x7c00。之后Os获得CPU的控制权。

Os的Interface:面向外设的中断和IO，面向应用的系统调用和异常来提供相应的一些功能。

中断:来源于外设，处理时间为异步，帮助Os完成相应的分时调度，时钟外设会定期的产生中断，把控制权交给Os，让Os完成这个切换的过程。中断分两部分完成:硬件设置中断标记[CPU初始化]，对相应中断产生中断号，Os根据中断号能找到对应的处理历程。软件(Os)会保存被打断的执行现场，然后根据中断号对应的地址跳转执行，清除中断标记后，恢复现场。

系统调用:应用程序主动向Os发出服务请求。例如C的printf()执行时会触发系统调用write()。

应用程序调用系统调用时，Os会从用户态转换为内核态，从而使得控制权从应用程序交到Os。当应用程序发出系统调用之后，切换到kernel中执行时，需要切换堆栈，函数调用则不需要。

Os对内存分层体系:会将主存抽象成连续的逻辑地址空间，保护独立的地址空间，共享访问相同的内存，虚拟化更多的地址空间。

访问内存流程:当CPU要执行某条指令时，ALU需要指令的内容，发出带有逻辑地址的请求，MMU会查找逻辑地址的映射表中是否存在对应的物理地址，如果MMU没有，就会去Memory的Map中找。