2 操作系统结构

系统调用（System Call）和API的区别

- 系统调用一般提供给C/C++语言；API可以提供给更多语言；
- 系统调用运行在内核态，API运行在用户态；
- 系统调用轻量级，API功能完善；
- 系统调用大多实现在操作系统内部，API大多实现在操作系统外部。

系统调用的接口在用户态，但是系统调用的实现在内核态。

系统调用会有编号，以方便在内核态进行对应处理。

系统调用的类别

- 进程控制（Process control）
- 文件管理（File management）
- 设备管理（Device management）
- 信息维护（Information maintenance）
- 信息传递（Communication）
- 保护（Protection）

> [*Example*]
>
> ```
> # include <stdio.h>
> int main() {
>     printf("Hello, World!\n");
>  return 0;
> }
> ```
>
> `printf()`是C语言的API，内部实际上是系统调用`write()`的包装。
>
> 如果位于Linux内核态，一般使用的是`printk()`命令。

操作系统设计方法

- **分层设计**：分成若干层次，每个层次都有各自的功能；第0层（底层）是硬件，第n层（顶层）是用户接口；中间层是操作系统需要实现的。
- **微内核**：微内核内只包含操作系统必须拥有的功能；因此其容易扩展、容易添加新命令、更可靠、更安全；但是可能会带来额外的性能开销（需要频繁从用户态切换到内核态）。（文件系统、驱动程序都能在微内核之外）
- **模块化编程**（可以加载的内核模块）：用于补充操作系统功能，可增删模块。
- **混合方法**：采用上面三种方法的结合。（现在大部分都是采用混合方法）

链接（Linkers）：程序在运行的时候，可能调用了其他文件（如头文件），需要将当前程序与其他文件进行链接。

- 静态链接：一个程序运行的时候，直接建立链接；优点：稳定；缺点：如果有多个程序执行，链接相同文件，文件会被加载多次，导致内存浪费。
- 动态链接：为了解决文件被加载多次，需要写一个DLL文件，声明哪些程序可以共享一个头文件。有点：不需要重复加载某些头文件，内存空间节省，效率更高；缺点：需要写很多DLL文件。

加载（Loaders）：程序在运行的时候，可能调用了其他文件（如头文件），需要将其他文件的内容加载到内存里以供使用。

- 静态加载：一个程序运行的时候，将这个程序全部加载进内存。

- 动态加载：一个程序运行的时候，只将当前需要用到的内容加载进内存，内存的内容会不断替换，以保证空余内存。

为什么一个应用会有不同操作系统的版本？——因为操作系统实现的不同，接口也不同，导致应用的一些代码也不相同。