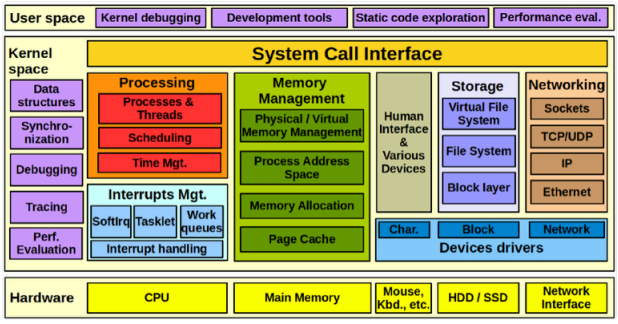
第08章：异常控制流

操作系统：唯一的使命就是帮助程序运行

视频解说

导读

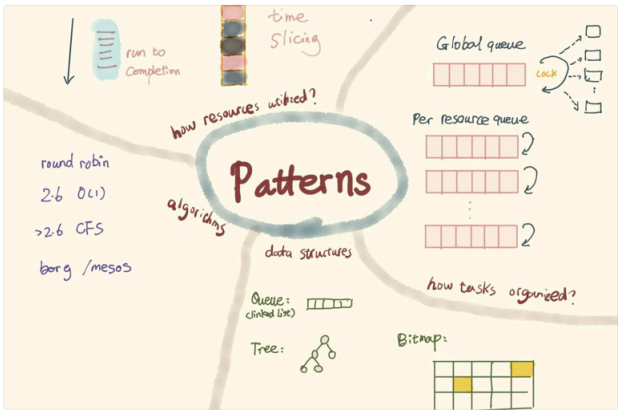
提示：本章题目是“异常控制流 = Exceptional Control Flow”，实际内容是进程，系统调用，异常，信号等，它们与操作系统（以及系统编程）之间都有着密切的联系，后续的几个章节，例如虚拟内存，系统I/O，网络编程等也都与操作系统（以及系统编程）有着密不可分的联系，这就意味着你在学习本章的时候应该顺带着学习操作系统（以及系统编程）相关的术语和基础知识，能够从高层角度理解一些重要概念：Process Management/Scheduling，Memory Management，File System，Interrupts，Device drivers，Networking，IPC ... 推荐学习Linux操作系统。



图片来源：佐治亚理工学院，Linux内核编程，Pierre Olivier

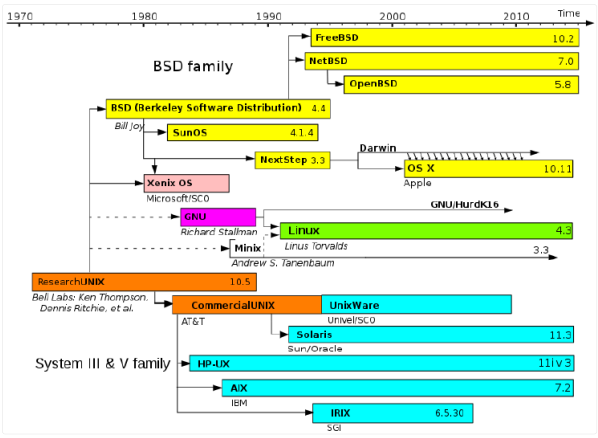
商用操作系统的具体实现一般都比较复杂，比如你可以看看Linux进程管理的 *task_struct*，很多决策都依赖于这个结构体中的内容，像是调度器（Scheduler ）的实现就相当复杂（注意：不仅仅是操作系统会有调度器，很多控制系统都有，比如基站），需要考虑诸多议题，例如公平（Fair）、优先级（Nice）、抢占(Preemption)、效率（Efficiency）、扩展能力 (scalability，比如从单核到多核，支持NUMA等），在移动设备上甚至还要考虑能效（Energy，比如ARM big.LITTLE，Tickless Kernel）... 本质上是一个工程问题！

根据业务需求有不同的 pattern 或 trade-off，以Linux为例，我们设计调度器的时候要考虑以下这些问题：



图片来源：知乎，陈天（https://zhuanlan.zhihu.com/p/33389178）

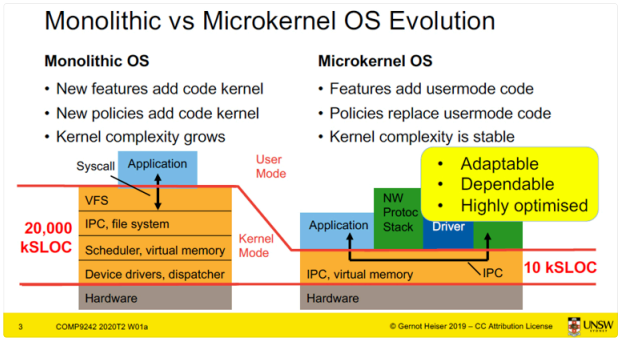
操作系统的历史：类Unix操作系统的发展与演进（参考：Unix操作系统 - 历史回忆录）



图片来源：佐治亚理工学院，Linux内核编程，Pierre Olivier

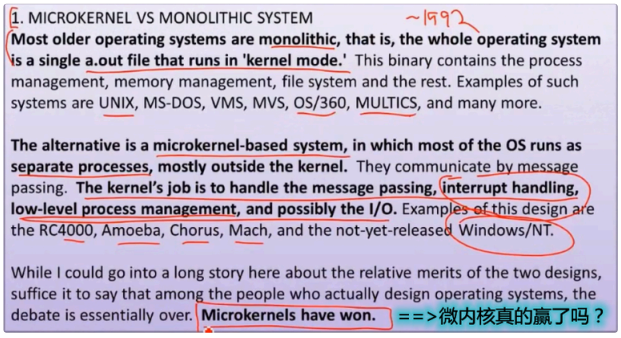
操作系统（内核）分类：Monolithic kernel (宏) vs. Micro kernel (微) vs. Hybrid kernel (混)

视频：宏内核 vs 微内核 (Monolithic-kernel vs Micro-kernel)



来源：新南威尔士大学，操作系统进阶课程

历史：Andrew Tanenbaum教授与Linus Torvalds关于宏内核与微内核的争论邮件



Andrew S. Tanenbaum教授（Minix作者）关于宏内核和微内核的意见

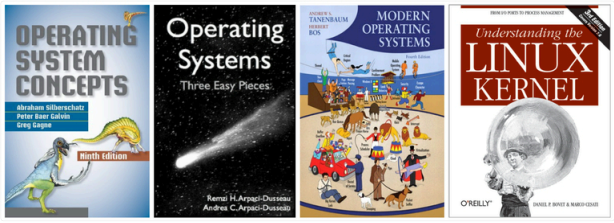
混合内核的例子：苹果的很多技术来自Steve Jobs于1985 年离开Apple Computer之后创立的 NeXT公司，后者的主力产品就是NeXTSTEP 操作系统，它以CMU Mach为基础，整合了BSD4.3作为uerspace server，后来苹果公司收购了NeXTSTEP ，并将其技术发扬光大，演化成了 XNU (核心) / Darwin (操作系统)，其开源代码请参考：https://github.com/apple/darwin-xnu，造就了今天iOS / macOS 所使用的关键技术。

学习方式

CMU教授的视频课程 - Lecture14：异常 & 进程

CMU教授的视频课程 - Lecture15：信号 & 非局部跳转

操作系统的书籍：附录中可以直接下载电子书，如果是自学的话，建议从下图中的第二本书开始学，全名 Operating Systems: Three Easy Pieces，简称OSTEP，它是威斯康星大学的研究生教材，分成虚拟化、并发性、持久化，三方面来讲，其实写的很入门，完全能当本科教材或者自学，每一个主题都是从历史沿革来讲，最初什么方法，如何实现的（真的是实际实现），解决了什么问题，有什么缺点，针对这些缺点人们提出了哪些方法来改进。



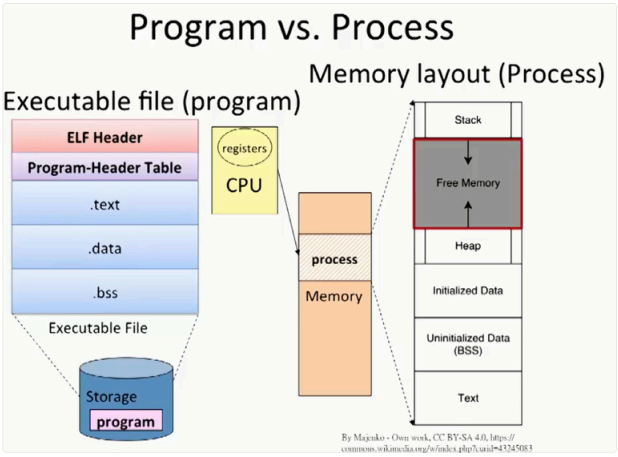
操作系统相关的书籍推荐

提示：书本总是最后才出现在我们手中的（并且有可能你拿到的时候就已经是过时的了），Linux这样的现代系统是"活着的" 操作系统，比如它需要考虑如何支持 SMP （Symmetric multiprocessing），支持虚拟化技术 (Xen 和 KVM)、支持容器化 (Container) 的能力，支持实时性（Real-Time），等等，建议可以前往：https://www.kernel.org/doc/ 阅读和查找你感兴趣的主题，同时可以关注一些世界级的Linux大会（比如：Linux基金会的官方网站：https://events.linuxfoundation.org/，里面汇集了众多开发者大会链接，许多优秀的内核开发者的演讲视频都可以在Youtube上找到，当然都是免费的），还可以定期浏览一些很好的新闻网站，比如：https://lwn.net/，https://www.linuxjournal.com/

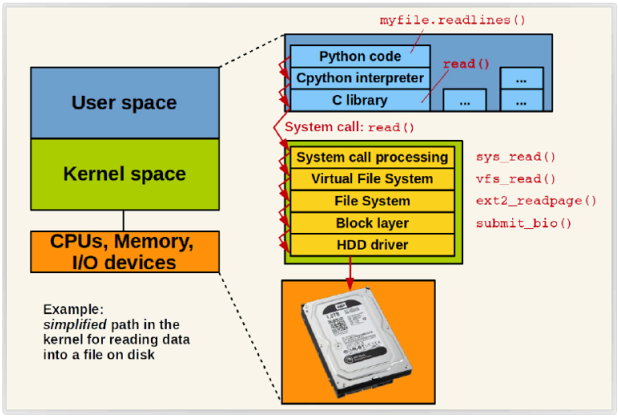
世界一流大学的线上课程（尽量选择最新的）也可以拿来参考学习（参见"延伸阅读"部分）

重点解读

程序 vs 进程

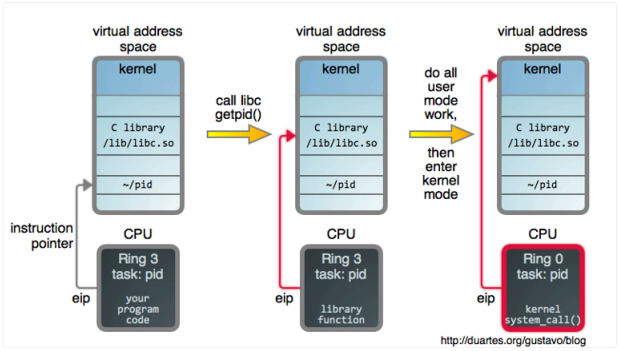


用户空间 vs 内核空间



图片来源：佐治亚理工学院，Linux内核编程，Pierre Olivier

这里给大家举一个例子，假设你的程序会调用getpid()这个系统调用（其中会陷入**中断/异常**）：

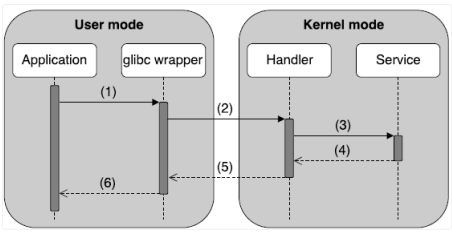


System Calls Make the World Go Round

库函数 vs 系统调用

像是**fork**这样的系统调用（fork没有参数，一切都继承自父进程【懒惰】，更加诡异的是，它返回两次）早在Unix第1版就已经存在，你看： Unix第1版的手册（第2章：系统调用） ，这样算下来这个系统调用已经存在长达50年了，最终fork的思想也被 Linux 继承和发扬光大，你知道Linux是怎么实现进程/线程的么？

实际上在Linux中fork()是一个封装了底层**clone**()系统调用的库函数（man 2 fork），你可以使用**ltrace**来追踪库函数调用，使用**strace**命令来追踪系统调用，起码你需要对用户模式和内核模式也要有基本的概念。



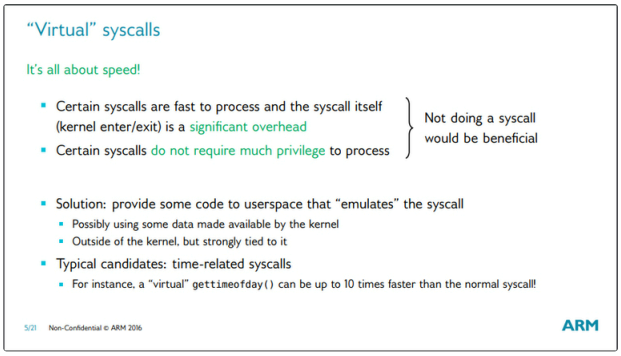
系统调用的时序图（图中假设通过glibc，当然也可以绕过它直接进行系统调用）

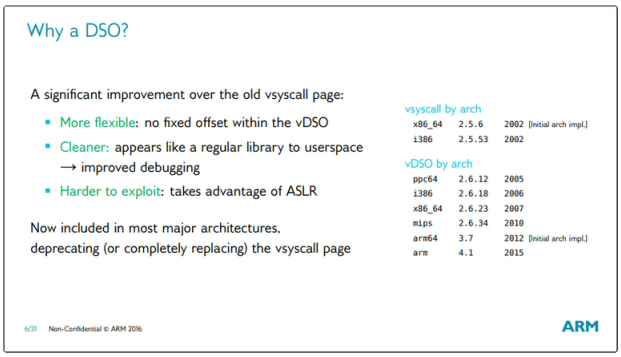
```c
// 你最熟悉的程序
#include <stdio.h>
int main(void) {
    printf("hello, world!\n");
    return 0;
}

// 用gcc编译之后，通过ltrace来追踪库函数调用，strace来追踪系统调用

// 稍微改写一下，让我们更加接近底层一些：man syscall
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
int main() {
    return syscall(__NR_write, 1, "Hello, world!\n", 14);
}

// 真正的系统调用是sys_xxxx（通过ltrace -S可以看到）

// 感兴趣的同学可以参考Linux内核源码
// https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscall_64.c
// https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl
```

另外，初次接触fork()函数的同学，可能会被"printf"输出多少次的问题弄得比较晕乎，类似的题目：<UNIX环境高级编程> 系列视频课程，进程环境和进程控制：04:02 ~ 07:18，在学习进程和进程创建相关知识后，你应该要能够摸清其中的来龙去脉。

虚拟系统调用 （Virtual syscall），虚拟动态共享对象（VDSO）

如果你查看 cat /proc/self/maps，会发现vsyscall，vdso，可能会好奇它们是什么东西。





https://blog.linuxplumbersconf.org/2016/ocw/system/presentations/3711/original/LPC_vDSO.pdf

延伸阅读

- 印度理工学院：操作系统导论 (非常好的入门课) -- 有课程的英文逐字稿
- 麻省理工学院：操作系统导论 6.S081 (2020年) -- 视频搬运到B站了（搜6.S081即可）
- 加州伯克利分校：操作系统与系统编程 CS162 -- 视频搬运到B站了（搜CS162即可）
- 斯坦福大学：操作系统 CS140 -- 没有视频，可以自行下载教学投影片学习
- 浙江大学：操作系统 (2018年秋冬季) -- 没有视频，可以自行下载教学投影片学习
- 毕尔肯大学：操作系统原理 (CS-342)
- 本人拙作：《Unix环境高级编程》视频课程的《进程环境和进程控制》，《信号控制》：Unix环境高级编程

Previous
第07章：链接

Next
第09章：虚拟内存