

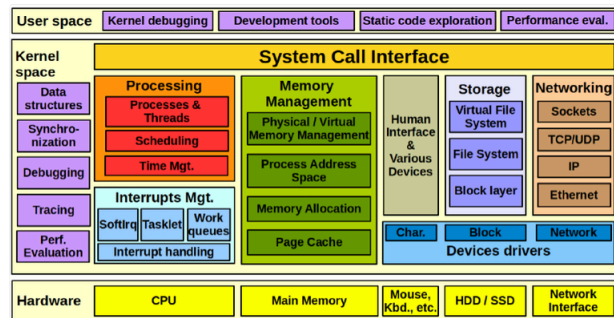
第08章：异常控制流

操作系统：唯一的使命就是帮助程序运行

视频解说

导读

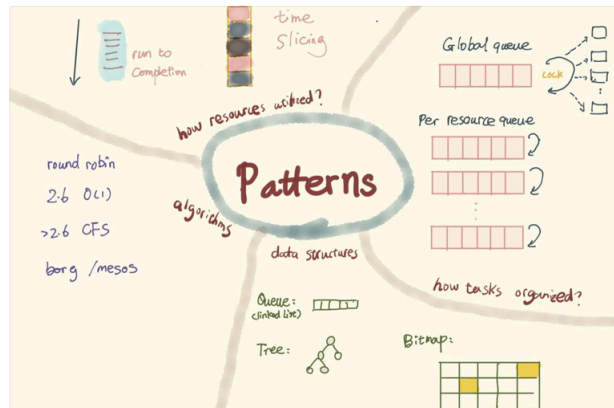
提示：本章题目是“异常控制流 = Exceptional Control Flow”，实际内容是进程，系统调用，异常，信号等，它们与操作系统（以及系统编程）之间都有着密切的联系，后续的几个章节，例如虚拟内存，系统I/O，网络编程等也都与操作系统（以及系统编程）有着密不可分的联系，这就意味着你在学习本章的时候应该顺带着学习操作系统（以及系统编程）相关的术语和基础知识，能够从高层角度理解一些重要概念：Process Management/Scheduling, Memory Management, File System, Interrupts, Device drivers, Networking,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

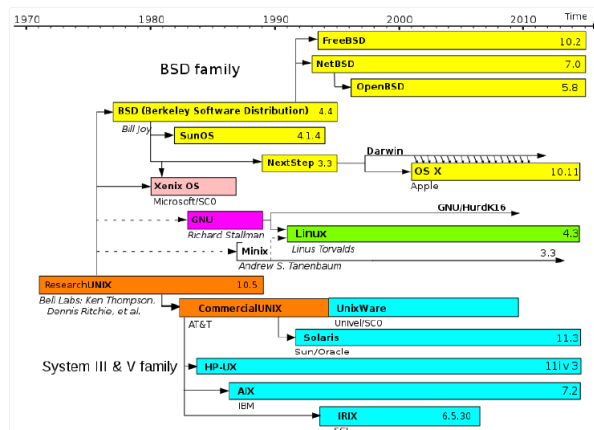
商用操作系统的具体实现一般都比较复杂，比如你可以看看Linux进程管理的 `task_struct`，很多决策都依赖于这个结构体中的内容，像是调度器（Scheduler）的实现就相当复杂（注意：不仅仅是操作系统会有调度器，很多控制系统都有，比如基站），需要考虑诸多议题，例如公平（Fair）、优先级（Nice）、抢占（Preemption）、效率（Efficiency）、扩展能力（scalability，比如从单核到多核，支持NUMA等），在移动设备上甚至还要考虑能效（Energy，比如ARM big.LITTLE，Tickless Kernel）... 本质上是一个工程问题！

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](#)）

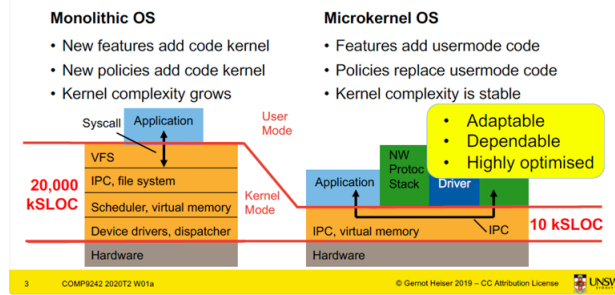


图片来源：佐治亚理工学院，Linux内核编程，Pierre Olivier

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 \(Monolithic-kernel vs Micro-kernel\)](#)

Monolithic vs Microkernel OS Evolution



来源：新南威尔士大学，操作系统进阶课程

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

1. MICROKERNEL VS MONOLITHIC SYSTEM

Most older operating systems are monolithic, that is, the whole operating system is a single a.out file that runs in 'kernel mode.' This binary contains the process management, memory management, file system and the rest. Examples of such systems are UNIX, MS-DOS, VMS, MVS, OS/360, MULTICS, and many more.

The alternative is a microkernel-based system, in which most of the OS runs as separate processes, mostly outside the kernel. They communicate by message passing. The kernel's job is to handle the message passing, interrupt handling, low-level process management, and possibly the I/O. Examples of this design are the RC4000, Amoeba, Chorus, Mach, and the not-yet-released Windows/NT.

While I could go into a long story here about the relative merits of the two designs, suffice it to say that among the people who actually design operating systems, the debate is essentially over. Microkernels have won. ==> 微内核真的赢了吗？

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

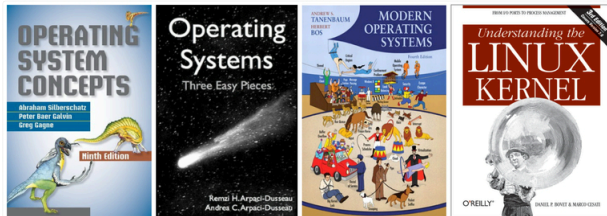
混合内核的例子：苹果的很多技术来自Steve Jobs于1985年离开Apple Computer之后创立的NeXT公司，后者的主力产品就是NeXTSTEP操作系统，它以CMU Mach为基础，整合了BSD4.3作为userspace server，后来苹果公司收购了NeXTSTEP，并将其技术发扬光大，演化成了XNU(核心)/ Darwin(操作系统)，其开源代码请参考：<https://github.com/apple/darwin-xnu>，造就了今天iOS / macOS所使用的关键技术。

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。



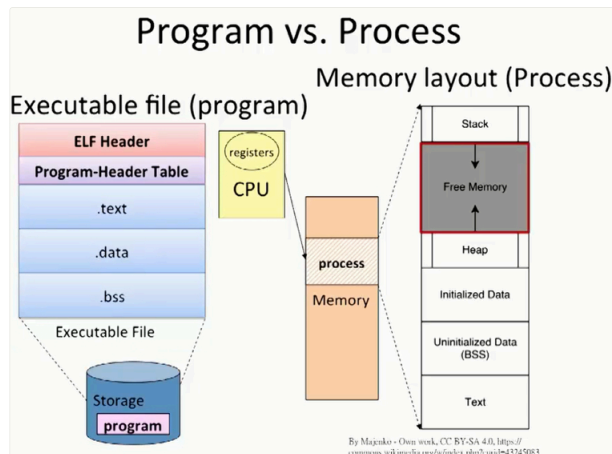
操作系统相关的书籍推荐

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专题，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/>

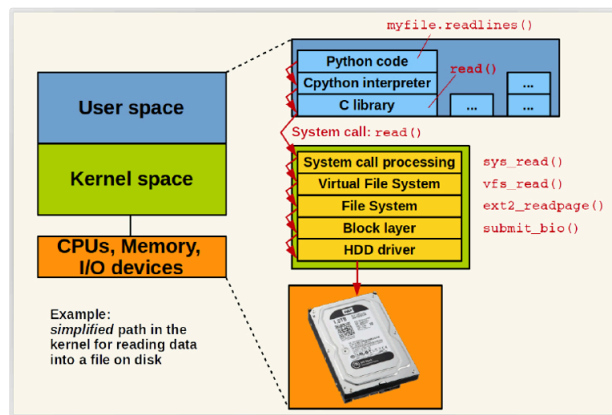
世界一流大学的线上课程（尽量选择最新的）也可以拿来参考学习（参见“延伸阅读”部分）

重点解读

程序 vs 进程

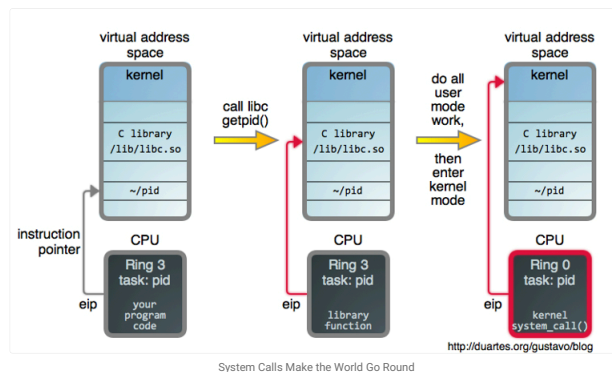


用户空间 vs 内核空间



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）：

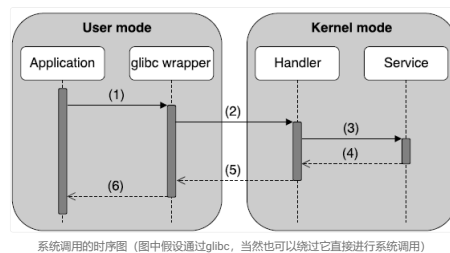


System Calls Make the World Go Round

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。



```
// 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“print”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。

“Virtual” syscalls

It's all about speed!

- Certain syscalls are fast to process and the syscall itself (kernel enter/exit) is a **significant overhead**
- Certain syscalls **do not require much privilege** to process

} Not doing a syscall would be beneficial

- Solution: provide some code to userspace that “emulates” the syscall
 - Possibly using some data made available by the kernel
 - Outside of the kernel, but strongly tied to it
- Typical candidates: time-related syscalls
 - For instance, a “virtual” gettimeofday() can be up to 10 times faster than the normal syscall!

5/21 Non-Confidential © ARM 2016 **ARM**

Why a DSO?

A significant improvement over the old vsyscall page:

- More flexible**: no fixed offset within the vDSO
- Cleaner**: appears like a regular library to userspace
 - improved debugging
- Harder to exploit**: takes advantage of ASLR

Now included in most major architectures, deprecating (or completely replacing) the vsyscall page

vsyscall by arch			
x86_64	2.5.6	2002	[initial arch impl.]
i386	2.5.53	2002	
vDSO by arch			
ppc64	2.6.12	2005	
i386	2.6.18	2006	
x86_64	2.6.23	2007	
mips	2.6.34	2010	
arm64	3.7	2012	[initial arch impl.]
arm	4.1	2015	

6/21 Non-Confidential © ARM 2016 **ARM**

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

Last updated 5 minutes ago