我是一个进程

我听说我的祖先们生活在专用计算机里，一生只帮助人类做一件事情，比说微积分运算了、人口统计了、生成密码、甚至通过织布机印花！
如果你想在这些专用“计算机”上干点别的事儿，例如安装个游戏玩玩，那是绝对不可能的，除非你把它拆掉，然后建一个全新的机器。而我这些祖先们勉强可以称为“程序”。
后来有个叫冯诺依曼的人，非常了不起，他提出了“存储程序”的思想，
并且把计算机分为五大部件：运算器、控制器、存储器、输入设备、输出设备。
各种各样不同功能的程序写好以后，和程序使用的数据一起存放在计算机的存储器中，即“存储程序”；然后，计算机按照存储的程序逐条取出指令加以分析，并执行指令所规定的操作。
这样一来，原来的专用计算机变成了通用的计算机，不管你是计算导弹弹道的，模拟核爆炸的，还是计算个人所得税的，统统都可以在一台机器上运行，
我就是其中的一员：专门计算员工的薪水。
进程的诞生我所在的计算机是个批处理系统，每次上机时，我和其他程序都排好队，
一个接一个的进入内存运行。
每个月末是发薪日，我都要运行一次，这样我每月都能见一次CPU阿甘，
这个沉默寡言，但是跑的非常快的家伙。
我知道内存看阿甘不顺眼，还告了它一状，说他一遇到IO操作的时候，就歇着喝茶，从来不管不问内存和硬盘的忙的要死的惨境。（码农翻身注：参见《CPU阿甘》和《CPU阿甘之烦恼》）

其实我倒是觉得挺好，这时候正好和阿甘海阔天空的聊天，他阅程序无数，
知道很多内部消息，每一个字节都清清楚楚，和他聊天实在是爽。

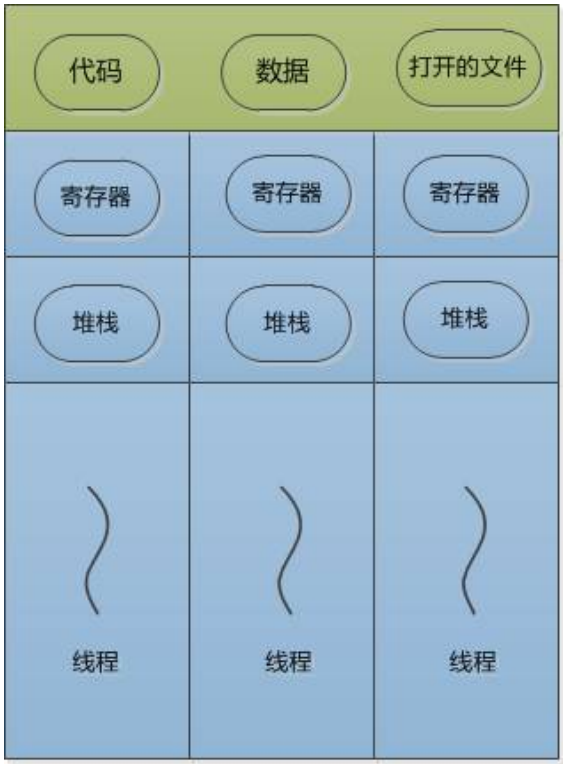
又到了月末发薪水的时候，我刚一进入内存，便看到这么一个公告：

公告　为了创建和谐社会，促进效率和公平，
充分发挥每一个人的能力，经系统党委慎重研究决定：本系统自即日起，正式从“批处理系统”转为“多道程序系统”，希望各部门通力配合，一起完成切换工作。系统党委
xxxx年xx月xx日
我正想着啥是多道程序系统，阿甘便打电话给内存要我的指令开始运行了。
和之前一样，运行到了第13869123行，这是个IO指令，我欢天喜地的准备和阿甘开聊了。
阿甘说：哥们，准备保存现场吧，我要切换到另外一个程序来运行啦!

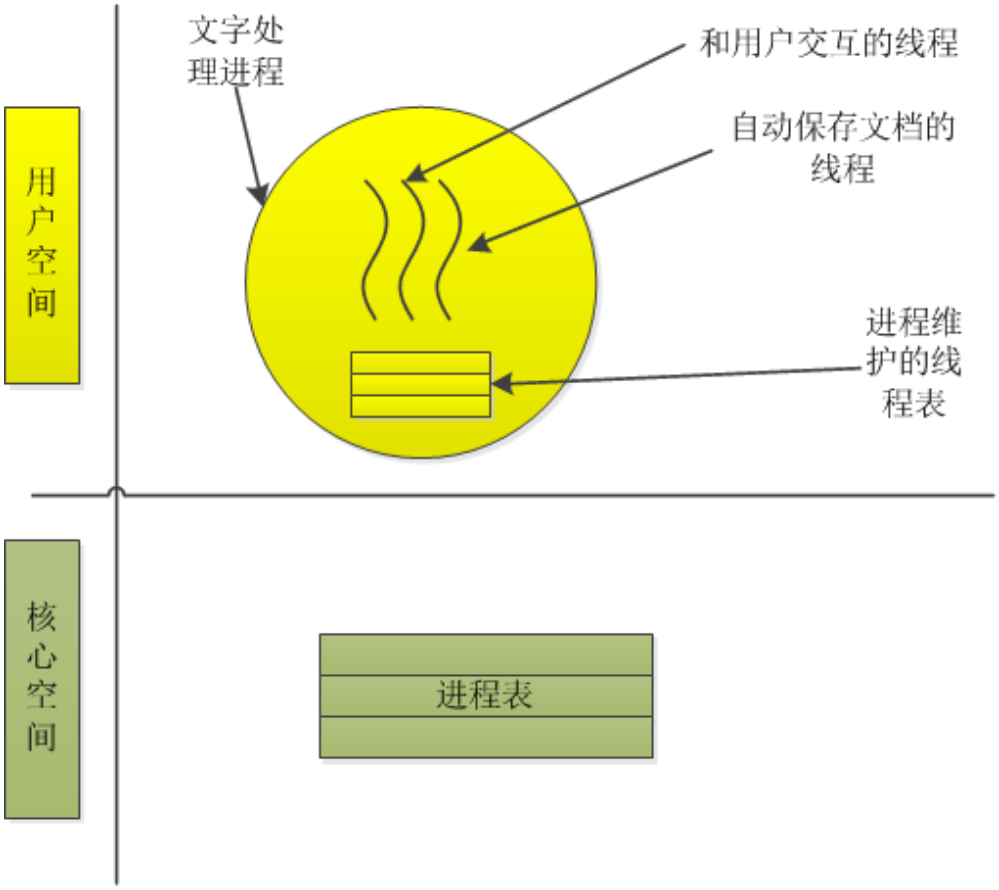
“啊？ 我这正运行着呢！ 咱们不喝茶了？
“喝啥茶啊，马上另外一个程序就来了!”
" 那我什么时候回来再见你？ ” 我问道。
“等这个IO指令完成，然后操作系统老大会再给你机会运行的。”
“那谁来记住我当前正在运行第13869123行？ 还有刚把两个数据从内存装载到了你的寄存器，
就是那个EAX, EBX , 你一切换岂不都丢了？ ” 我有点着急。
阿甘说：“所以要暂时保存起来啊，
不仅仅是这些，还有你的那些函数在调用过程中形成的栈帧和栈顶，
我这里用寄存器EBP和ESP维护着，
都得保存起来。”（码农翻身注：参见《CPU阿甘之函数调用的秘密》）

“还有” 阿甘接着说，“你打开的文件句柄，你的程序段和数据段的地址，
你已经使用CPU的时间，等待CPU的时间。。。。。。
以及其他好多好多的东西，统统都要保存下来。”
我瞪大了眼睛：“这也太麻烦了吧，原来我只需要关心我的指令和数据，
现在还得整这么多稀奇古怪的东西”
“没办法，这就叫做上下文切换， 把你的工作现场保存好，这样下一次运行的时候才能恢复啊。
对了，老大给你们统一起了一个新的名称：**进程**！ 刚才那些需要保存的东西叫做叫做**进程控制块**(Processing Control Block， **PCB**)，”
我想了想，这个名字还挺贴切的，
一个真正进行的程序！ 只是这个正在进行的程序随时可以被打断啊。
我只好保存好上下文，撤出CPU，
回到内存里歇着去了，与此同时另外一个程序开始占据CPU运行。
其实我这个程序，奥，不对，我这个进程被放到一个阻塞队列里，等到IO的数据来了以后，又
被赶到了就绪队列中， 最后才有机会再次运行，再次见到CPU阿甘。（码农翻身： 进程的就绪，
阻塞，运行这三个状态的转换和《我是一个线程》中描述的非常类似）
阿甘从我的PCB中取出各种保存的信息，恢复了运行时现场， 可是忙活了好一阵，
没办法，这就是程序切换必须要付出的代价。
我有点同情阿甘了，
从此以后，他很难再悠闲和和我们海阔天空，每时每刻都处于高速的奔跑中。
得益于阿甘的高速度， 虽然在同一时刻只有一个程序在运行，
但是有很多程序在短时间内不断的切换， 在外界看来， 似乎多个程序在同时执行。
尤其是那些速度超慢的人类， 他们开着电脑一边听歌，一边上网，一边QQ，很是自在，
理所当然的认为这些程序就是同时在运行。 岂不知阿甘是让音乐播放器上运行几十毫秒，
然后打断，让浏览器进程运行几十毫秒，再打断，让QQ也运行几十毫秒，如此循环往复。
唉，阿甘真是能者多劳啊，
这个计算机系统也算是达到了我们党委的目标：兼顾了效率和公平。线程有了进程就万事大吉
了吗？ 人类的欲望是无止境的，很快就出现了新情况，
举个例子来说吧，我有一个兄弟，是个文字处理软件， 他和我不一样， 他有界面，
人类在用的时候能看到， 这实在是很幸福， 不像我总是在背后默默工作，几乎无人知晓。
这哥们有个智能的小功能，就是在人类编辑文档的时候能自动保存，
防止辛辛苦苦敲的文字由于断电什么的丢掉。
可是这个功能导致了人类的抱怨，
原因很简单，自动保存文字是和IO打交道，那硬盘有多慢你也知道，
这个时候整个进程就被挂起了， 给人类的感觉就是： 程序死了，键盘和鼠标不响应了!
无法继续输入文字， 但是过一会儿就好了。
并且这种假死一会儿就会出现一次（每当自动保存的时候）， 让人不胜其烦。
系统党委研究了很久， 他们当然可以用两个进程来解决问题， 一个进程负责和用户交互，
另外一个进程负责自动保存，

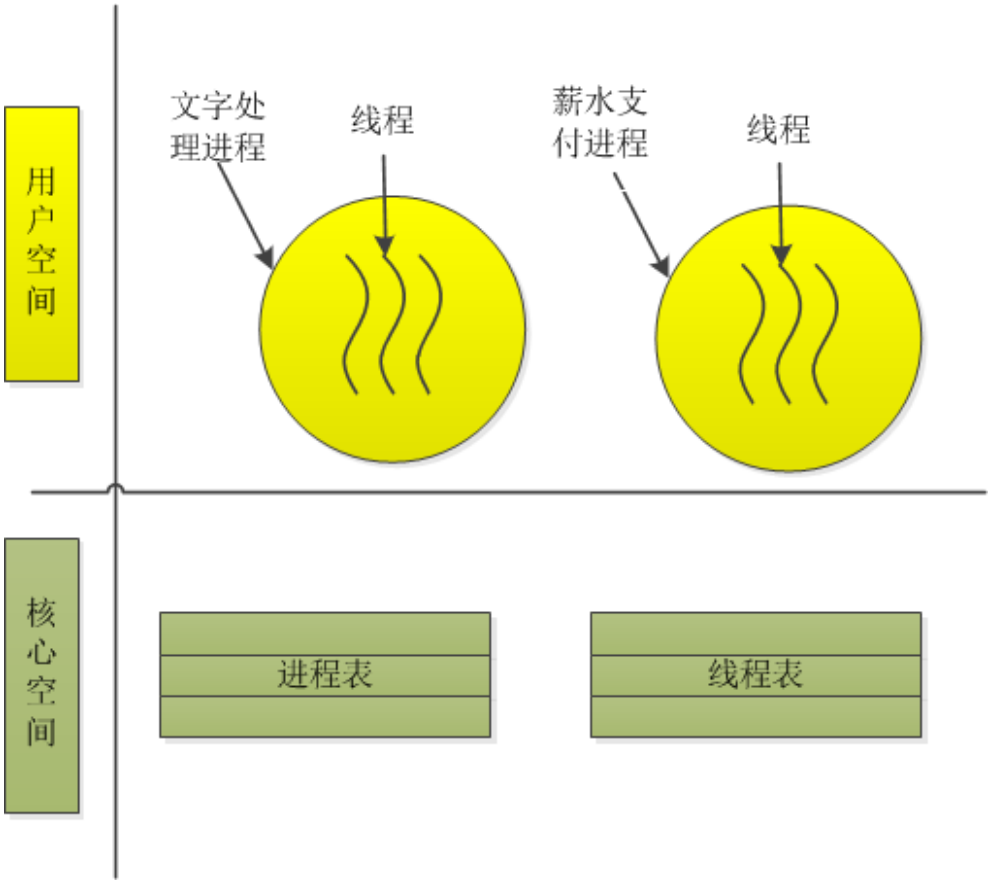
但是，这两个进程之间完全是独立的，每个人都有自己的一亩三分地（地址空间），
完全互不知晓，进程之间通信的开销实在是太大，
他们没有办法高效的操作那同一份文档数据。后来还是劳模阿甘想出了一招：可以采用多进程的伟大思想啊!
把一个进程当成一个资源的容器，让里边运行几个轻量级的进程，就叫**线程**吧，
这些线程共享进程的所有资源，例如地址空间，全局变量，文件资源等等。
但是每个线程也有自己独特的部分，那就是要记住自己运行到哪一行指令了，
有自己的函数调用堆栈，自己的状态等等，
总而言之，就是为了能像切换进程那样切换线程。



拿我那个哥们的情况来说，一个进程保存着文档的数据，进程中有两个线程，
一个负责和用户交互，
另外一个专门负责定时的自动保存，IO导致的阻塞就不会影响另外一个了。
注意，这两个线程都能访问进程的所有东西，他们两个要小心，不要发起冲突才好 --
这是人类程序员要做的事情了，不归我们管。争吵阿甘的建议被采纳了，
其实这几乎是唯一的解决问题方式了，但是由谁来管理引起了激烈争吵。
系统党委有一波人坚持要在用户空间实现线程，
换通俗的话说就是让那些进程在自个儿内部去管理线程，
他们的理由也很充分：你们自己实现了线程，可以自己定制自己的调度算法，多灵活啊；
所有的线程切换都在进程内完成，不用请求我们操作系统内核来处理，效率多高啊；
况且你们可以在那些内核不支持线程的操作系统中运行，
移植性多好啊。

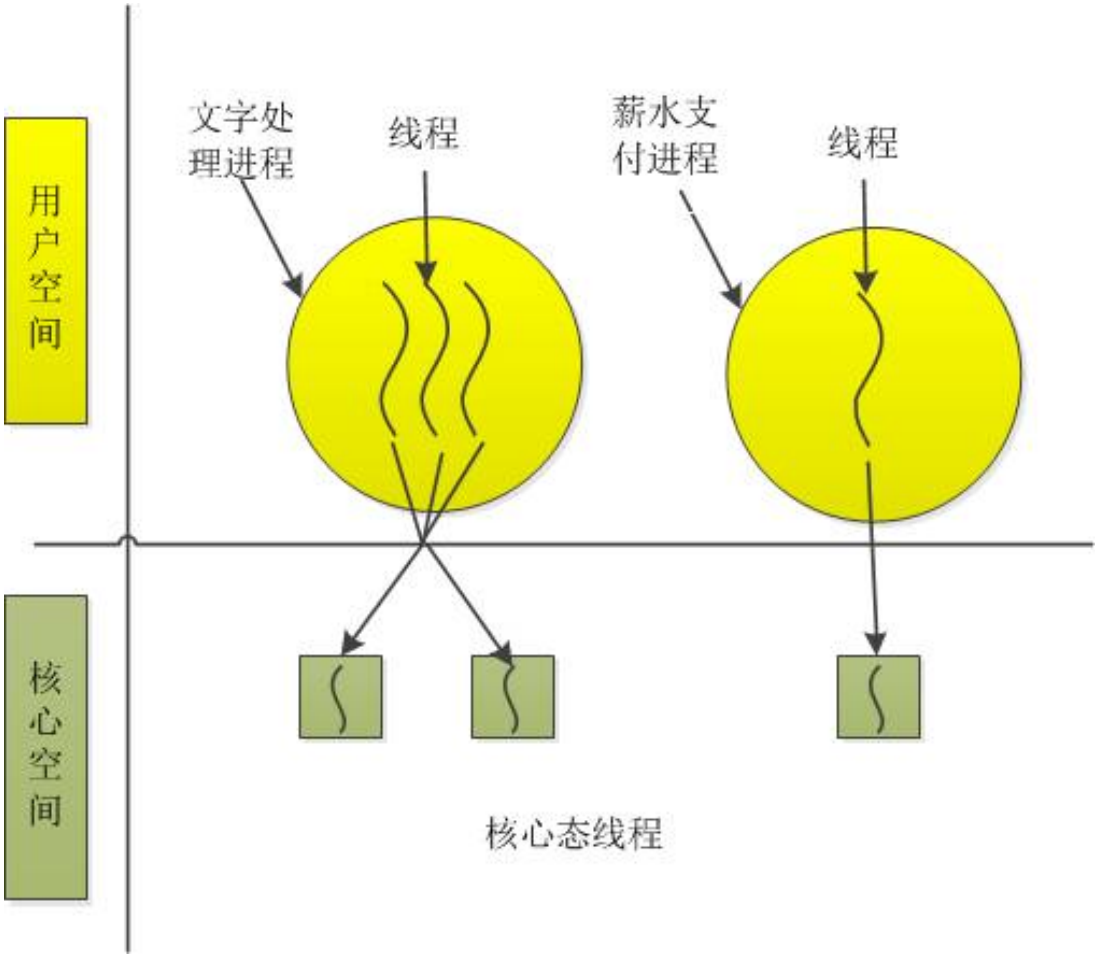


我们清楚的知道这是内核想做甩手掌柜， 因为他们选择性的忽略了一个致命的问题:
如果由我们实现线程，则操作系统内核还是认为我们只是一个进程而已，对里边的线程一无所知，对进程的调度还是以进程为最小单位。
一旦出现阻塞的系统调用，不仅仅阻塞那个线程，还会阻塞整个进程!
例如文字处理器那个进程， 如果负责定时保存的线程发起了IO调用，
内核会认为，这是由进程发起的，于是就把整个进程给挂起了，
虽然和用户交互的进程还是可以运行，也被强制的随着进程挂起来，不响应了， 这多么悲催啊，
又回到了老问题上去了。
所以我们坚决不能答应， 我们则一致的要求: 在内核中实现线程！
内核需要知道进程中线程的存在，内核需要维护线程表，并且负责调度!



党委的人傲慢的说： 你们不嫌累吗， 每次创建一个线程都得通过我们内核,多慢啊。
我们说： 只有这样， 一个线程的IO系统调用才不会阻塞我们整个进程啊,
你们完全可以选择同一个进程的另外一个线程去执行。
双发僵持不下， 最后只好妥协， 那就是： 混合着实现吧。
用户空间的进程可以创建线程（用户线程）， 内核也会创建线程（内核线程），

用户线程映射到内核线程上。



问题基本解决了，但也带来了新的问题，我们的系统也变的越来越复杂， 尤其是进程之间的通信和线程之间的同步， 会那些程序员们带来无穷无尽的烦恼， 这是后话了， 有机会下次再说吧。

注： 本文的插图来源于《现代操作系统》和《操作系统概念》（恐龙书）这两本书， 我重画了一下。 对操作系统感兴趣的同学可以看看这两本书。