[十年大佬讲述，操作系统内存管理(图文详解) - 知乎](https://zhuanlan.zhihu.com/p/495709005)

[操作系统——段式存储管理、段页式存储管理 - 王陸 - 博客园](https://www.cnblogs.com/wkfvawl/p/11733057.html)（这个非常好的告诉你为什么要分段为什么要分页）

[内存分段与分页 | 一个工匠](https://www.yigegongjiang.com/2022/memory/)（主要明白cpu和memory的过程）

首先，为什么要分段？

如果你写程序的时候，没有逻辑地址这个概念，写的所有地址都是物理地址，那么会发生以下几个问题：

1. 两个进程之间可以之间相互访问并且修改数据，也就是没有隔离。
2. 无法动态重定向进程的内存，什么意思，就是你在程序中写的物理地址变成了一个定值，这是不对的。
3. 随着不断分配不同区域的进程，会有越来越多的外存碎片，频繁的搬运合并空闲块显然不太合适

因此，主要就是为了抽象出来这个内存地址，让不同进程以为自己都运行在某个内存区间，大家都是从地址 0 开始到 某个地址。首先是，将进程划分为几个段，按照不同功能（代码段，堆，栈，数据段等），然后每个段能对应到物理内存中同样大小的段。

为什么还要分页？

前面说过，因为你每个段的大小是不同的，随着分配的段越来越多，外存碎片越来越多，那么有没有什么办法，让我的逻辑内存，映射到不连续的物理内存呢？这就是分页，也就是在分段的基础上，每个逻辑段里面，再切分成好几个逻辑页，一个逻辑页大概4kb，很小。

A diagram of a computer

AI-generated content may be incorrect.

段页式的图，可以看到，一个进程，在逻辑内存上分段，在实际物理内存上是分页的。

A diagram of a computer

AI-generated content may be incorrect.

段表和页表都是存放在内存中的，有部分会被cache在cpu cache中。

我们需要注意，每个进程有一个段表，多个页表。

1. 段号S + 段表始址 F <= F + M ? 如果是，继续执行，否则越界
2. 段号S \* 段表项长度 + 段表始址 F 得到段表项的地址（物理地址），然后查询段表项的页表在物理内存中的块号，以及这个段的页表长度（也就是有多少块），也就是我们通过段表，获得了这个段所对应的页表的地址（物理地址）。
3. 通过页表地址+页号\*页表项长度，我们得到了段号S，页号P的这个逻辑页的物理块号是多少（其实块就是物理页，他们大小是一样的），并且通过页内偏移得到这个对应的物理地址。

三次访存，第一次通过段号得到这个段所对应的页表地址，第二次通过页号得到这个逻辑页的物理页地址，第三次则是通过物理页地址，实际获得真正的数据。