关于进程的讨论

- 进程的分类
  - 系统进程
  - 用户进程
  - 前台进程
  - 后台进程
  - CPU密集型进程
  - I/O密集型进程
- 进程层次结构

UNIX进程家族树：init为根
Windows：地位相同

啊，所以这是呢进程之间的关系。

进程与程序的区别

- 进程更能准确刻画并发，而程序不能
- 程序是静态的，进程是动态的
- 进程有生命周期的，有诞生有消亡，短暂的；而程序是相对长久的
- 一个程序可对应多个进程
- 进程具有创建其他进程的功能

生活中类比例子

那么这就是一个，进程的一个日常生活中的例子

进程地址空间

操作系统会给每一个进程都分配了一个地址空间
怎么理解这个，这句话？怎么理解这样一个场景？

先看一段程序

```
int myval;
int main(int argc, char *argv[])
{
    myval = atoi(argv[1]);
    while (1)
        printf("myval is %d, loc 0x%lx\n",
            myval, (long) &myval);
}
```

同时运行两个myval程序

输出？

- Myval 7
- Myval 8

那么输出的结果 是什么样子呢？ 我们来看一下

Acknowledgement: Prof. Yuanyuan Zhou at UCSD

输出结果如下

```
liyc@romteam:~/temp$ ./myval 7
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
myval is 7, loc 0x60104c
```

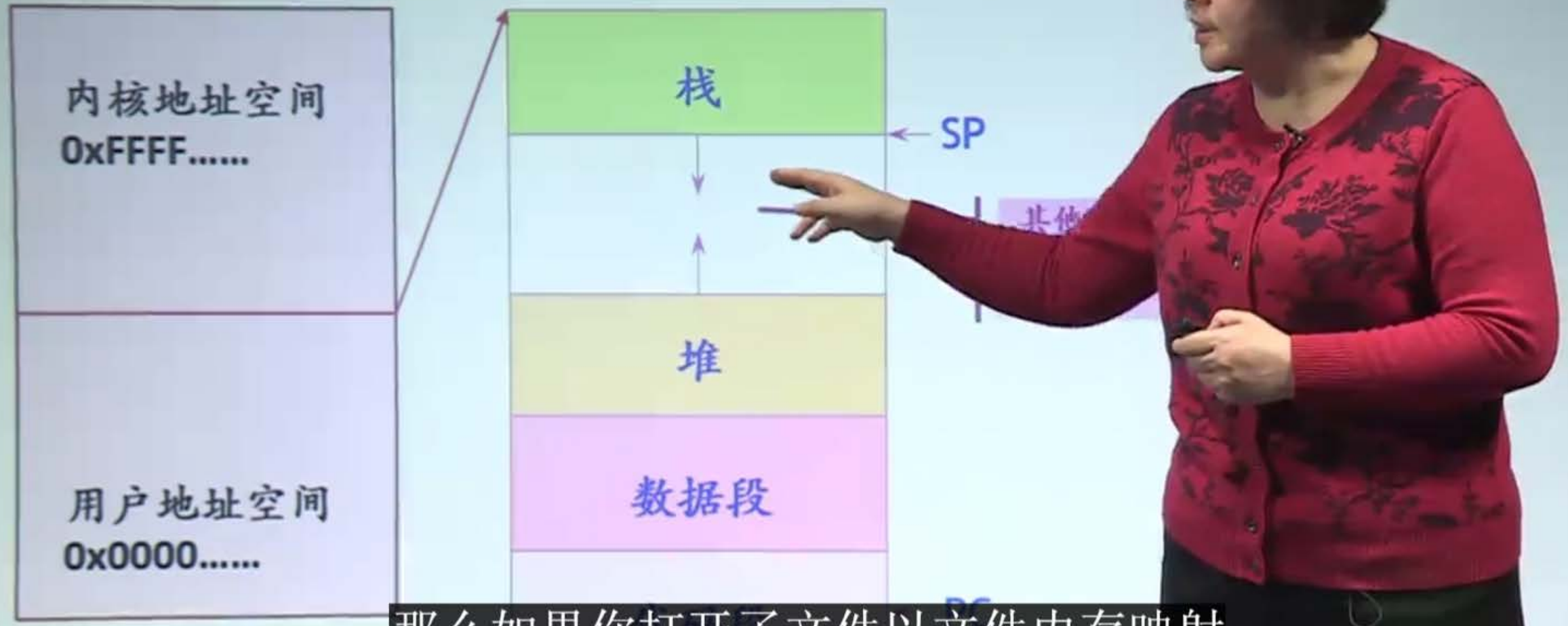
```
liyc@romteam: ~/tem
liyc@romteam:~/temp$ ./myval 8
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
myval is 8, loc 0x60104c
```

变量myval
的值不同，
但地址相同
?

我们也可以说它是一个相对地址，或者是逻辑地址

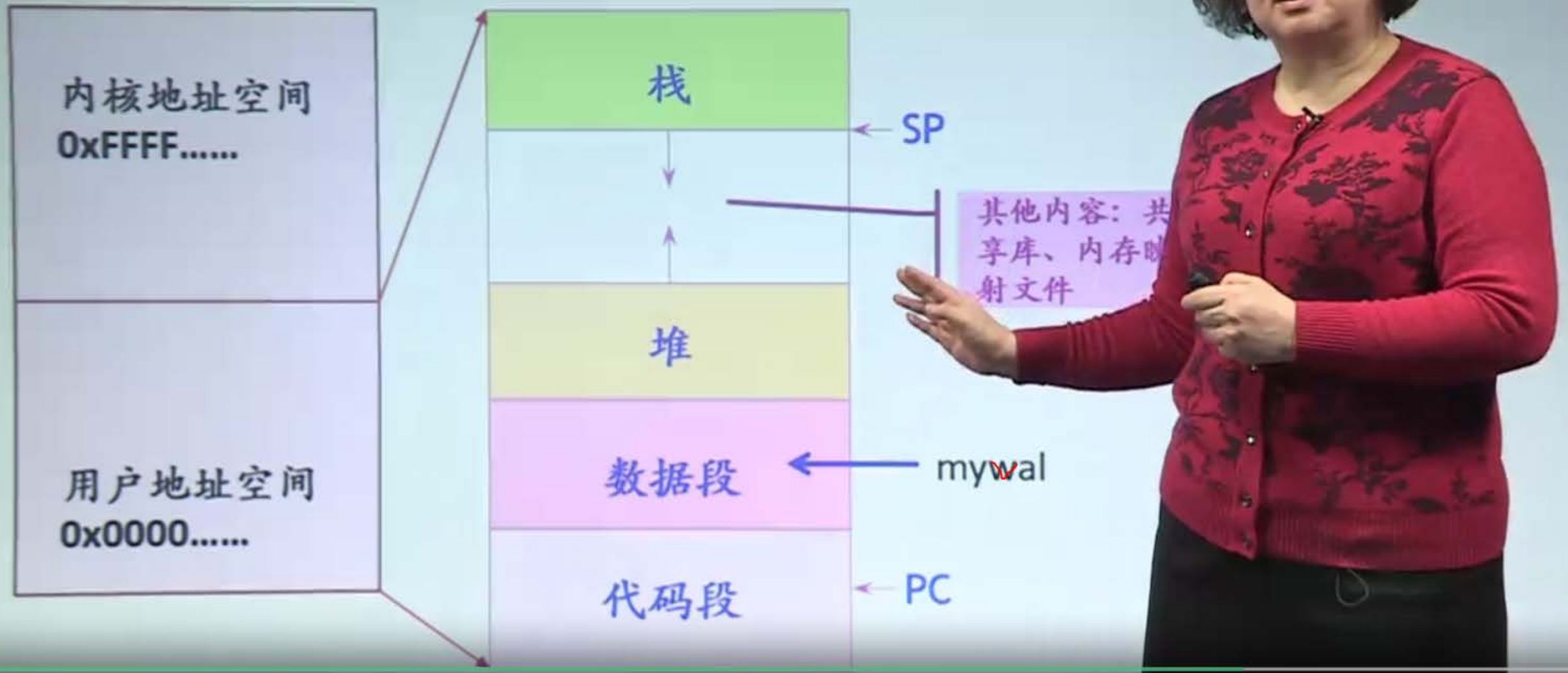
Acknowledgement: Prof. Yuanyuan Zhou at UCSD

进程地址空间



那么如果你打开了文件以文件内存映射
文件的方式来使用这个文件的话呢，那么也用到这些空间

进程地址空间


内核地址空间
0xFFFF……
用户地址空间
0x0000……
栈
SP
堆
数据段
mywal
代码段
PC
其他内容：共
享库、内存映
射文件

进程映像（IMAGE）

对进程执行活动全过程的静态描述

由进程地址空间内容、硬件寄存器内容及与该进程相关的内核数据结构、内核栈组成

- 用户相关：进程地址空间（包括代码段、数据段、堆和栈、共享库……）
- 寄存器相关：程序计数器、指令寄存器、程序状寄存器、栈指针、通用寄存器等的值
- 内核相关：
  - 静态部分：PCB及各种资源数据结构
  - 动态部分：内核栈（不同进程在进入内核后使用不同的内核栈）

上下文（CONTEXT）切换

- 将CPU硬件状态从一个进程换到另一个进程的过程称为上下文切换

- 进程运行时，其硬件状态保存在CPU上的寄存器中

寄存器：程序计数器、程序状态寄存器、栈指针、通用寄存器、其他控制寄存器的值

- 进程不运行时，这些寄存器的值保存在进程控制块PCB中；当操作系统要运行一个新的进程时，将PCB中的相关值送到对应的寄存器中

0:00 下面呢，我们来深入探讨一下进程的这些概念 下面我们看一下进程的分类。 第一种分类呢 实际上把进程分成了系统进程和用户进程 系统进程呢，是操作系统为了管理一些资源而设计的进程 它的主要特点是优先级比较高 相对于用户进程而言 系统进程会优先被调度上 CPU 去执行 因为它完成了一些关键的工作 第二类呢，我们称之为前台和后台进程 前台进程实际上就是和用户直接交互的这样一些进程 用户敲键盘、动鼠标，那么都是直接控制这些进程的 而后台进程呢，往往是操作系统在启动了以后，啊，初始化以后那么创建的一些进程。 那么这些进程 为用户来进行服务，比如说打印进程 当然了，还有一些应用进程，也在系统启动的时候呢被创建了 像比如说防火墙，还有一些电子邮件的接收，这样的一些进程 那么它在后台工作，然后呢发生了一些事件。 它们来接收这些事件 那么对于用户来讲呢，他所打交道的呢，是前台进程 另一种 进程的分类呢，实际上是考虑了进程的行为 有一些进程需要用到很多的 CPU 时间 比如说，在玩游戏的过程中，那么 有一些画面的渲染，那么是需要大量的计算 因此呢，我们把它称之为 CPU 密集型进程 那还有一类进程呢，可能经常需要输入、 输出、 读盘这样一些操作 那么这些进程呢，我们称之为 I/O 密集型 那么这两类 进程的区分呢，也是为了以后调度程序的选择做一些准备 操作系统中有很多的进程 创建了很多的进程，这些进程之间 它们的关系是否非常密切呢？我们来看一下 UNIX。 UNIX 呢它有一个概念，是一个进程家族 树这个概念。 也就是说，所有的 UNIX 进程都是在一个家族里头 那么这个家族树里头呢，有一个根。 这个根呢是一个 init 进程 是个 1 号进程。 这个 1 号进程呢，实际上是所有进程的一个祖宗 那么为什么要有这样一个树的概念呢？ 也就是说，在某些情况下，某一个进程它结束了 那么它的子孙进程，其实也必须全部的结束 那么它是有这样一个亲密的关系 那么这个关系呢，我们通常称之为一个层次结构 那在 Windows 当中呢，也是一个进程创建另一个进程 但是创建完另外一个进程之后呢，这两个进程的关系呢比较疏远 也就是它们的地位是平，相同的 啊，不存在这个说，一个进程结束，另外一个进程也必须结束，这样一个情况 啊，所以这是呢进程之间的关系。 下面 我们来看一下，进程和程序的区别。 这是两个比较容易混淆的概念 我们来看小明同学对这问题的理解 首先进程能够更准确地来刻画，并发环境、 并发程序的执行 而程序不可以。 程序通常是静态的 不改变它，它不会有变化 而进程呢是动态的，那么它会时时刻刻地去变化，因为它要往前执行 进程呢是有生命周期的 所谓生命周期就是，它有诞生、 有消亡 它是一个短暂的存在 而程序则是相对长久的 那么一个进程 可以对应多个，

一个程序 那么一个程序可以对应多个进程 比如说，一个编译程序 它编译不同的源代码，它就变成了多个编译进程 进程具有创建其它进程的功能，但是程序没有 那么对于 这两个概念的理解呢，我们可以通过日常生活中的例子 比如说，我们在教材上，我们经常举的一个例子是说，比如说一个食谱 怎么做菜？怎么做蛋糕？那么 有了食谱这就是有了程序。 那么做蛋糕的一个过程 或者是做菜的这么一个过程，炒菜这么一个过程，实际上就是一个进程 另外呢，我们 学了，啊，同学们选择了这个很多慕课课程 那么选择了操作系统啊，其它一些课程 每一门课程的学习，其实就是一个进程 那么这个进程过程当中，你要看 视频，你要做每周的小测验，最后还有一个考试 那么这个进程，那么有创建也有消亡 当 12 周、 13 周之后，通过了考试，拿到了证书。 那么这个进程呢，其实就结束了 那么每一个同学其实就是一个 CPU ，那么他 选了几门课，所以呢他要在 几门课之间来分配时间，就像 CPU 分配时间片一样。 那么这就是一个，进程的一个日常生活中的例子 下面我们对几个非常重要的概念呢进行讨论 刚才已经说过了，一个非常重要的概念是进程地址空间 操作系统会给每一个进程都分配了一个地址空间 怎么理解这个，这句话？怎么理解这样一个场景？ 我们来先看一段程序 但这个程序呢，实际上呢是从 命令行接收了参数，把这个参数赋给一个变量 myval 然后就是循环，来打印 这个变量的值，同时呢，把这个变量的位置 打印出来。 现在我们同时执行 两个 myval 程序。 那么也就是两个进程 那么这两个进程在执行，因为我们用了一个，大家可以看 用了一个循环，对吧，这个进程会一直在那运行下去 所以我们能看到一个，我们看到的结果。 那么如果我们运行这两个进程 一个我们用的是参数 17 ，另外一个我们用 8 来调用这个程序，来执行这个程序。 那么输出的结果 是什么样子呢？我们来看一下 当执行 myval 7 的时候 在屏幕上显示的是，myval 的这个变量，内容是 7 位置，也就是它的位置，地址是 60104C ，啊，这是 myval 7 那么同时 myval 8 也在执行。 在另外一个终端执行 我们看这几个不同的终端，分别执行 myval 程序。 那么这里显示的是 myval 的值呢是 8 ，这是参数 我们参数就是 8 ，是吧？命令行参数就是 8，这是对的。 然后呢 我们来看一下 myval 的地址。 myval 的地址呢 也是 60104C。 那我们就会 提出一个问题了。 变量 myval 的值是不一样的？ 为什么它的地址是相同的呢？这说明了什么问题呢？ 实际上 这也就是我们所说的，每个进程有自己相对独立地址空间的这样一个 结果。 两个进程实际上是两个地址空间 它们是，地址

空间是隔离的。 那么 不同的地址空间，它的地址 是内存地址吗？是实际的物理内存地址吗？ 不是。 它实际上是一个相对地址 如果支持虚存的系统当中，那么这个地址我们知道就是虚拟地址 我们也可以说它是一个相对地址，或者是逻辑地址 为了进一步来解释，我们来看看什么是进程地址空间 左边就是一个进程地址空间的一个情况 那么这个空间里头呢，其实操作系统会占一部分内容 所以呢，比如说我们说上半部分是操作系统内核的地址空间 而下面是用户地址空间，那么用户地址空间呢 都放些什么内容呢？这里头包括了用户执行的过程中所需要的一些代码 数据，一些临时的 变量放在堆里头。 还有呢 在运行过程中如果进行了过程调用 函数调用，需要用栈来传递参数 那么主体有这样一些内容 在进程运行过程中呢，还可能调一些共享库 因此呢，还有一些共享库放在这个位置 那么如果你打开了文件以文件内存映射 文件的方式来使用这个文件的话呢，那么也用到这些空间 那么这就是进程用户地址空间的内容 我们刚才所说的每个进程 my value 7 和 my value 8 这两个进程，每个进程都有这么一个地址空间 因此，myvalue 这个变量实际上 是在不同的地址空间里头的 相同的位置。 因此，我们看到了虚拟地址虽然是相同的 那么这个相同指的是对于这个地址空间 的位置，而不是指的在物理内存的位置 还有一个 容易引起混淆的概念呢，我们经常会说的叫进程映像 那么这个概念呢，指的是进程执行过程中 它的全过程的一个静态描述，可以把它看成是 在某一瞬间的进程的快照。 那么包括了什么内容呢，进程映像？ 包括了地址空间的内容 包括了硬件寄存器的内容，以及与 该进程相关的一些内核数据结构和内核栈 我们再强调一下，进程映像跟用户相关的呢，是这些地址空间内容，代码段、 数据段 堆、 栈，还有一些共享库等等。 那么跟寄存器相关的呢是这样一些 程序计数器、 指令寄存器、 程序状态字、 栈指针这样的寄存器的内容 还有和内核相关的 那么有，从静态的角度上讲，有 PCB 还有各种各样资源的数据结构 动态的角度上看呢，当一个 进程进入内核的时候呢，它需要用到的那堆，那一部分内核栈 所以你想找一个进程的所有的信息，需要在这几处 来发现这个进程的信息，所以我们说相当于一个快照，把这部分，这几部分内容把它 逻辑上归并在一起，这就叫做进程的映像 另外一个概念呢，叫做上下文切换 我们经常说这个 context ，context switch 那么它的概念实际上是说 CPU 的硬件状态从一个进程 换到另外一个进程的过程。 那么进程在运行的时候呢，它的硬件状态会 推送到若干 CPU 上的寄存器当中 这些寄存器我们都不陌生。 那

么但它不运行的时候 那么所有相关的硬件状态就不能放在这些寄存器中了，因为只有 一套寄存器。 那么这些内容呢就要把它保存在一个 安全的地方，那么这就是进程控制块的现场部分 那么如果一个操作系统又选中了一个新的进程 让它上 CPU 运行，那就要把保存在 PCB 当中的这些硬件状态把它推送到 相应的寄存器当中。 这个就完成了 上下文切换的一个过程