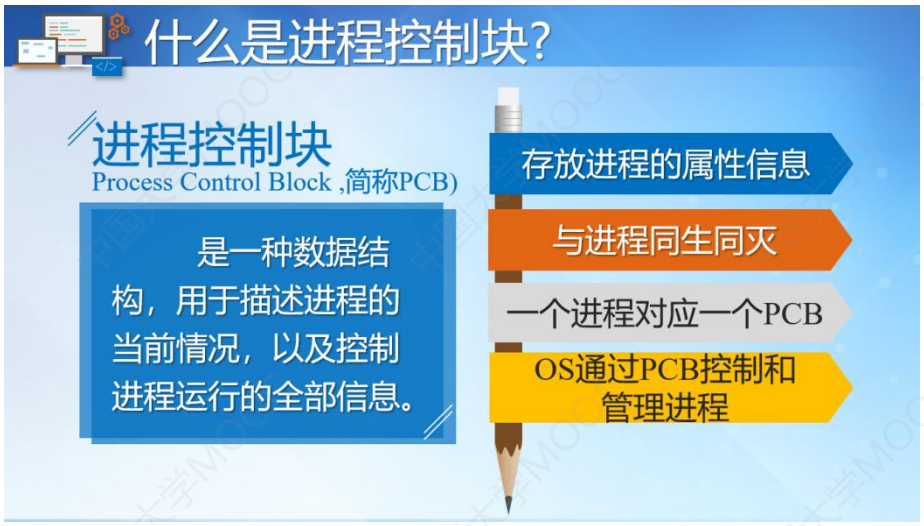
“进程控制块及 Linux 中的 task_struct 结构”导读宝典

上一讲多次提到进程，这一讲，我们讲解操作系统如何描述进程。
那么什么是进程控制块？

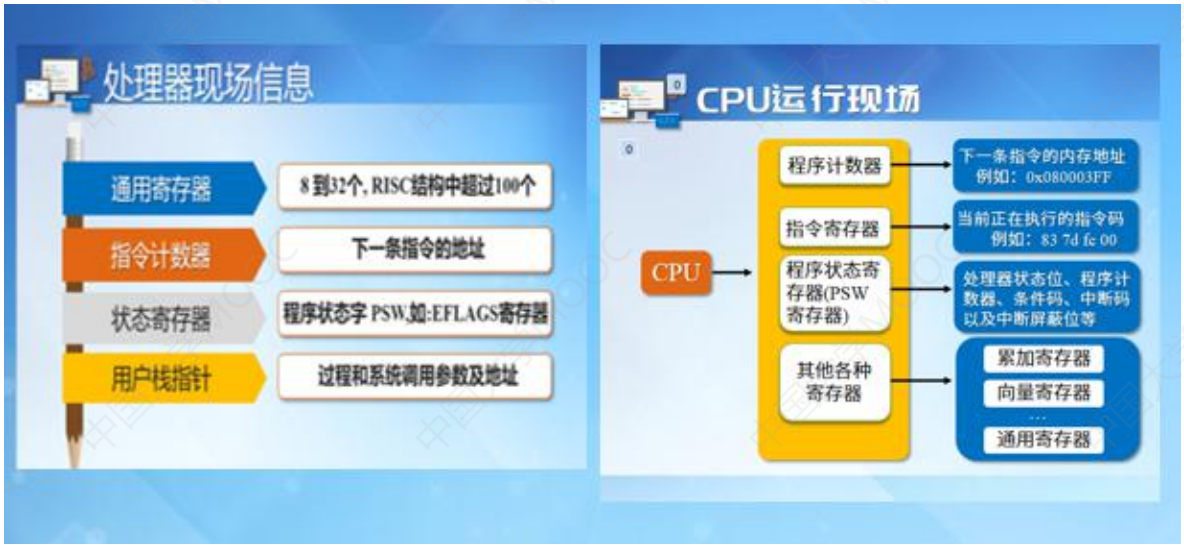


进程控制块(简称 PCB)是管理进程的数据结构，用它来记录进程的外部特征，描述进程的运动变化过程。OS 靠什么管进程，就是靠 PCB，相当于我们的身份证信息，但比身份证信息更全面，进程在执行过程中方方面面的信息都记录在其中。（查看哈工大李老师的视频，对于 PCB 的引入讲的比较透彻）

问题 1：类比一下，公安局是否相当于操作系统，通过身份证来对我们进行管理？

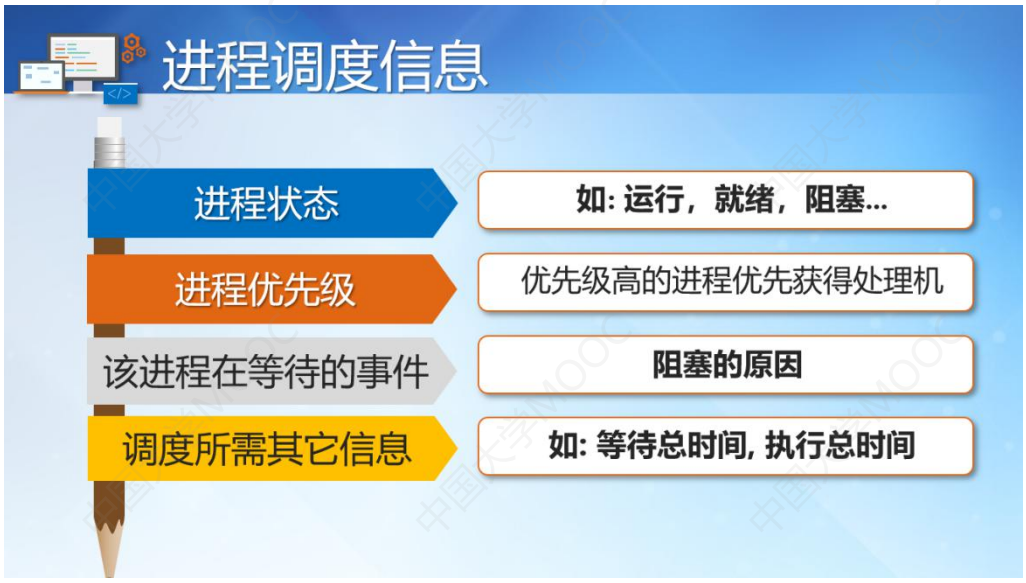
PCB 中的信息有哪些呢，非常多，为了易于描述起见，教科书中一般将其分为四类，包括进程标识信息，处理器现场信息，进程调度信息，进程控制信息。其中进程标识信息，唯一的标识进程，比如 PID，叫进程标识符，就相等于我们的身份证号，除此之外，还有进程组标识符等。

问题 2：进程标识信息除了 PID，其实还有很多的，uid，gid 等，就相等于我们身份证号，医疗卡号，社会保险号等等，这些信息起什么作用？除了标识的作用，是否是还用于安全？



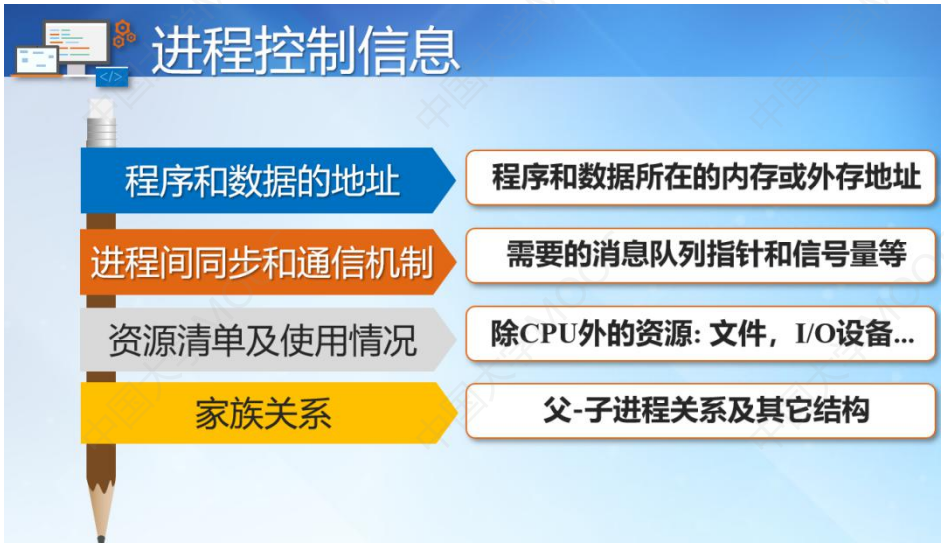
什么是处理器现场，就相当于我们开车过程出了事故的现场一样，那么，处理器的事故现场是什么，就是它本来好好的执行，被外设打断了，或者要从用户态陷入内核执行系统调用等等，这时必须保存处理器现场信息，这些信息通常包括通用寄存器，指令计数器，状态寄存器，用户栈指针等。保存现场是为了当事故处理完后恢复现场继续执行。上图中，左边是一般的处理器现场信息，右边是具体包含哪些寄存器。

问题：什么时候需要切换 CPU 现场，除了中断、异常和系统调用外，还有就是进程之间的切换，比如，从进程 A 切换到进程 B，但是归根结底还是上述三个场景之一，那么是中断、异常还是系统调用发生时进程才会发生切换？



什么是调度？就是就绪队列中的进程被选中到 CPU 上去运行，凭什么被选中，不是像公主抛绣球招婿，而是由很多因素决定的，比如进程状态，优先级，等待的事件，等待的总时间，执行的总时间等等。

问题：调度的例子在日常生活中比比皆是，比如航班调度，高铁调度等等，一个好的调度算法应该考虑什么？



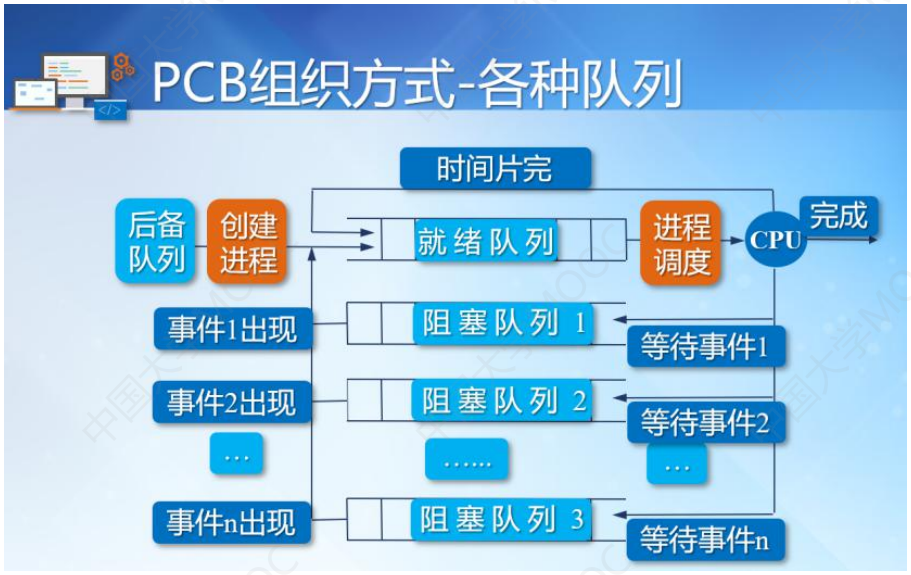
一个进程执行起来真不是一件简单的事，需要各种各样的控制信息，比如从哪里来，要到哪里去，也就是要将其程序从外存装入到内存，那么 PCB 中就需要有程序和数据的地址信息，还有进程间也可能需要打电话，这就是同步和通信机制，它们也需要带着干粮上路，也就是所使用的的资源，包括文件，I/O 设备等等，进程间也有七大姑八大姨的关系，这就是亲属关系以及相关数据结构等等。

问题：控制是什么意思？简单的说就是不要乱套。OS 就像一个总指挥一样协调各个进程各尽其职，各就各位，你可以联想日常生活中的很多例子。那么，如何做到控制，是否要掌握各种信息，除了这里列出的信息，你是否还能想到更多，比如，进程的创建，执行，撤销等等？

系统中可能有成百上千个进程，每个进程都有一个 PCB，怎么把它

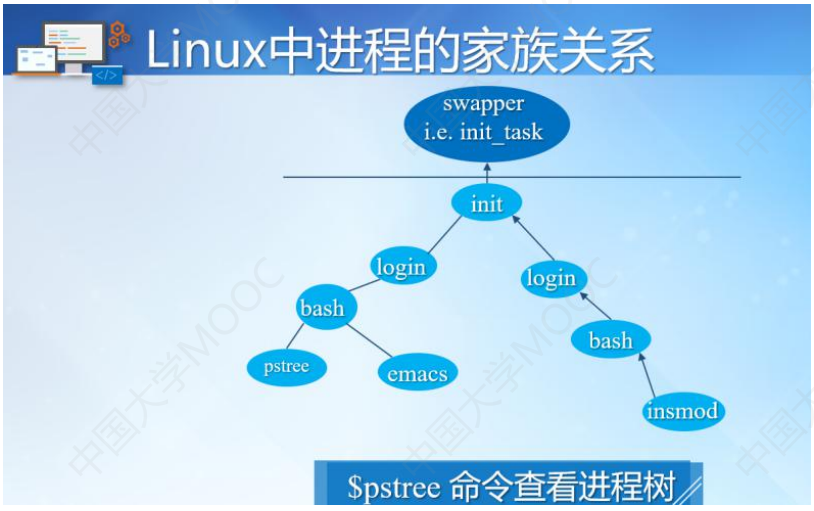
们组织起来。其组织方式无非就是数据结构中那些组织方式，比如线性表，链表，索引，哈希表，树型结构等。

问题：数据结构中学到的各种数据结构是不是排上用场了，那么对 PCB 组织的目的是什么？是不是为了快速的找到它（啦啦啦，原来才不过是查找算法），你觉得用哪种结构组织方式查找更快？



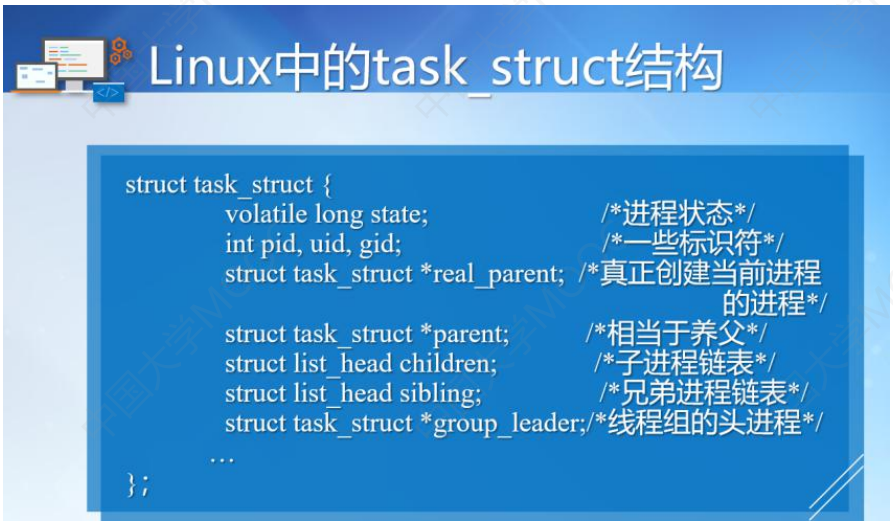
PCB 的组织方式与我们生活中的组织方式也类似，常见的就是排队。当一个进程创建时，其 PCB 就进入就绪队列等待被调度，当 CPU 空闲时，调度程序从就绪队列中选择一个进程投入运行，当一个进程时间片到时，它的 PCB 重新回到就绪队列，当一个进程在执行过程中进行 I/O 操作，就进入阻塞队列，阻塞队列可以根据等待事件的原因不止一个，当等待的事件完成后，该进程的 PCB 就进入就绪队列。你看，在每一种状态下，进程都回归到它自己的队列老老实实排队。

问题：看到队列你可能觉得是老朋友了，其实，你就想想你到银行办事，就这么简单，只是，你思考一下，队列用单链表还是双链表更好？



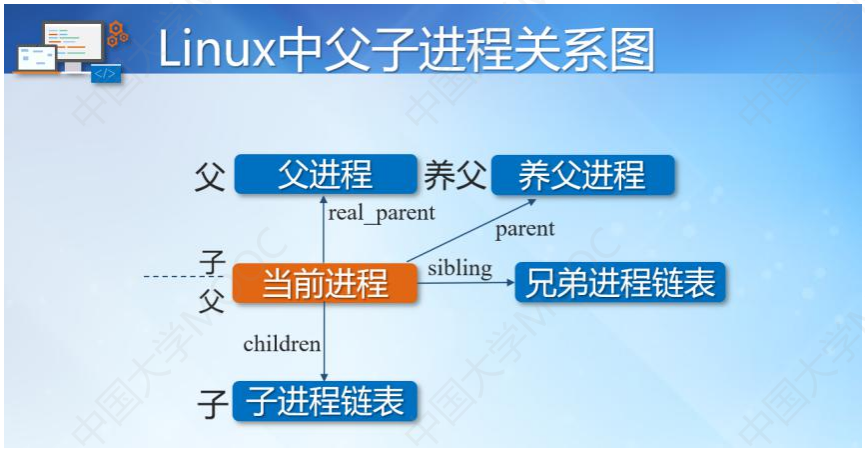
把进程想象成我们人类，有生有死就很好理解。进程就是一个动态的实体，它具有生命周期，系统中进程的生生死死随时发生。因此，操作系统对进程的描述模仿人类活动。一个进程不会平白无故的诞生，它总会有自己的父母。在 Linux 中，通过调用 fork 系统调用来创建一个新的进程。新创建的子进程同样也能执行 fork，所以，可以形成一颗完整的进程树，其中 init 进程是 1 号进程，是所有进程的祖先进程，也就是祖爷爷。如图是 Linux 系统启动以后形成的一棵树，可以通过 pstree 命令，查看自己机子上的进程树。

问题：查看李老师“多进程图像 1”8′左右，李老师说启动一个进程，执行一个任务，解决一个问题计算机就运行起来了，为什么说多进程图像重要？执行 pstree 命令查看你机子上的进程树。



Linux 中的 PCB 结构具体是什么呢，就叫 task_struct 结构，我们前面介绍的进程状态、标识符及亲属关系就都会存放在这个结构中，我们具体梳理一下：除了 1 号进程，没有进程是从石头缝蹦出来的，都有一个父亲，因为一个进程能生几个儿子，而儿子之间有兄弟关系，为了描述进程之间的父 / 子及兄弟关系，在进程的 PCB 中就要引入几个域。假设 P 表示一个进程，首先要有一个域描述它的父进程，就是 parent 域；其次，有一个域描述 P 的儿子，因为儿子不止一个，因此有一个域就指向老小（child）；最后，P 可能有哥哥弟弟，于是用一个域描述 P 的老哥（old sibling）,一个域描述 P 的老弟（younger sibling）

问题：原理 PCB 就是一个结构体，那么，这里的 struct list_head 是什么？它是 Linux 内核自己定义的一个双向链表，并且变为内核代码的基本类型，就像 int 类型一样在内核中使用，问题是，进程除了父亲，还有个养父到底干啥？

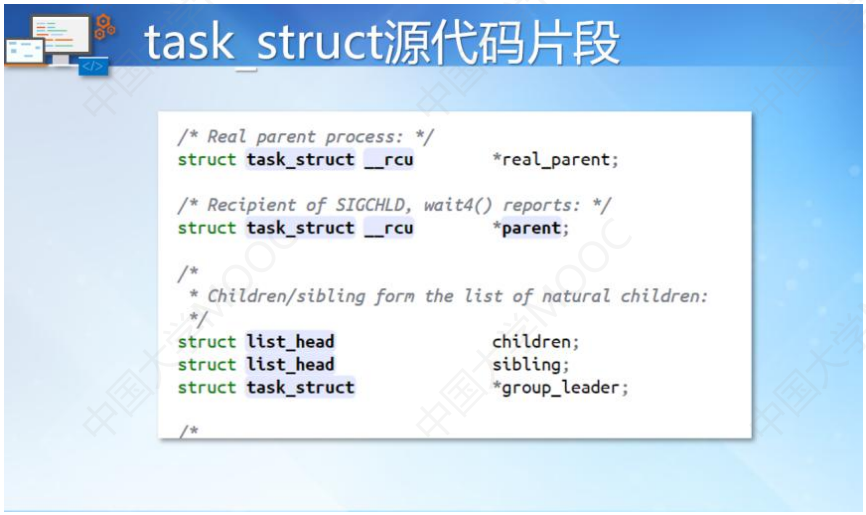


这里要说明一点是，一个进程可能有两个父亲，一个为亲生，一个为养父。为啥会有养父？一般应用中，都是父亲给儿子派活干，父亲等儿子干活结束后，把儿子的资源就收回了，但也可能出现儿子还在干活，结果父亲已经去世，儿子就成为孤儿了，于是就得给儿子重新找个养父，但大多数情况下，生父和养父是相同的。

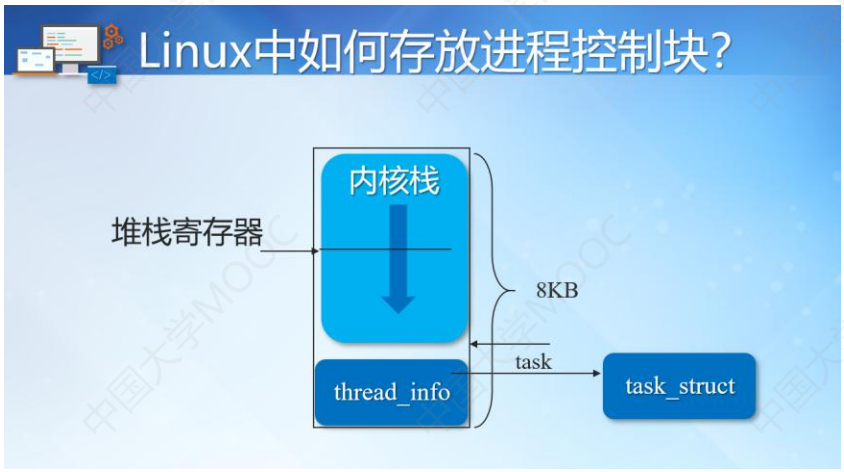
问题：Linux 中会出现僵死进程，查看资料，什么是僵死进程，系统中的僵死进程到底对系统有什么危害？

关于 task_struct 结构，我们进入 sched.h 头文件查看源代码（https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L629），从 629 行开始，会有身临其境的感觉，这里列出部分代码片段。其中 thread_info 结构保存进程 PCB 中频繁访问和需要快速访问的字段，内核依赖于该结构来获得当前进程的 PCB，而状态 state 就是一个无符号长整型，其中的修饰符 volatile 是 gcc 编译器的修饰符，有了这个修饰符，就是明确的告诉编译器，对该字段不要优化，从内存读取其值而不要从寄存器取。pid 是进程标识符，tgid 是什么呢，因为 Linux 内核支持内核级线程，每个进程中的所有线程就形成一个线程组，线程组就需要一个领导，这个 leader 的 ID，就是其所在进程的 pid。

问题：原来 PCB 的源代码长这个样子，问题 pid_t 又是啥类型，其实就是整型，像 volatile 这样的标识符从哪查，请查看 GCC 官方手册

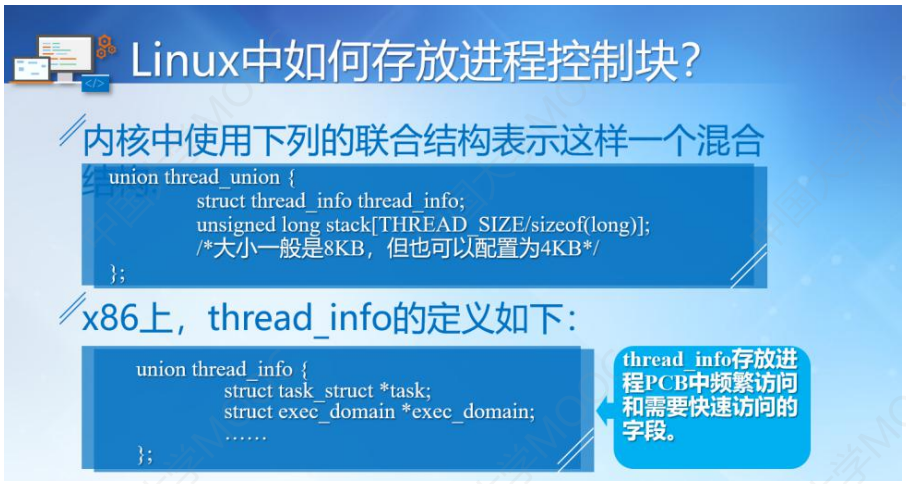


这是 task_struct 结构中进程家族关系的字段，可以看到，有亲手父亲，养父，孩子，兄弟还有组 leader 等。



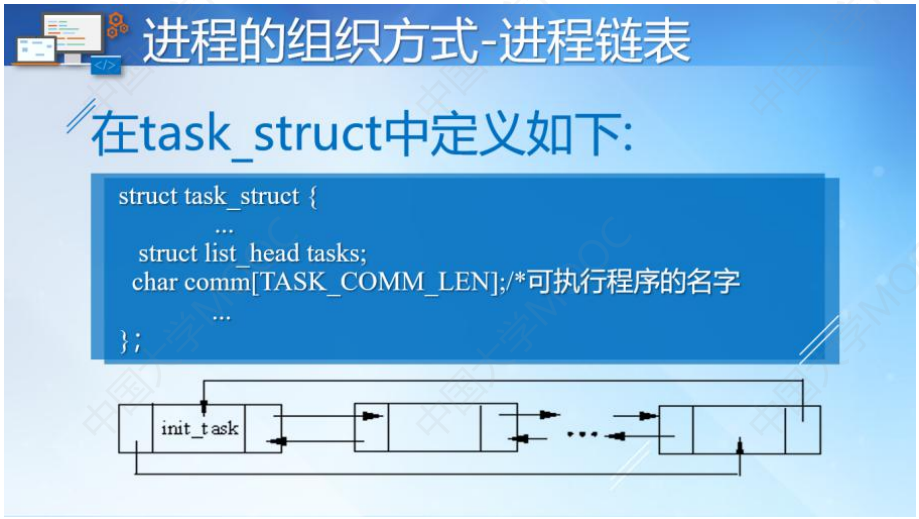
那么，进程控制块存放在内存的什么地方？Linux 在设计中发扬艰苦朴素的原则，为了节省空间，Linux 把内核栈和 PCB 的小数据结构 thread_info 放在一起，占用 8KB 的内存区，如下图所示。

问题：内核栈是什么东西？还记得用户态下每执行一个程序都有一个栈的，你说，我咋不知道，没有栈，函数的参数还有局部变量放在哪，函数的返回地址在哪找？进程执行系统调用进入内核后，就像在用户空间执行代码一样，也是一个函数调用一个函数，因此也需要栈的。这个栈怎么设计是很有讲究的，内核大神们就充分发挥 C 语言联合体结构的特点，让 PCB 与内核栈待在一起，你想想为啥？



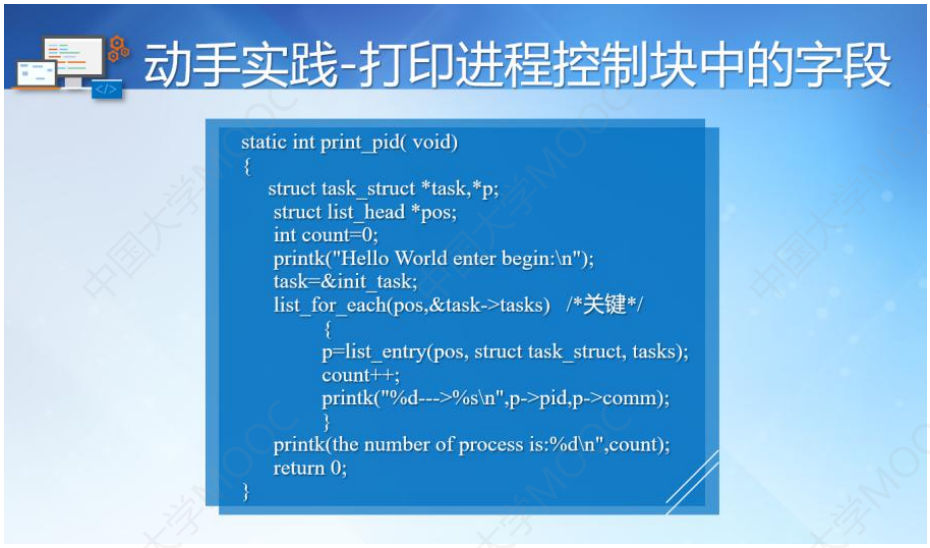
从这个联合体的定义可以看出，内核栈占 8KB 的内存区。 因为 thread_info 的第一个字段是 task_struct 结构。这 8K 的内核栈中，有一部分就让给 PCB 使用。

问题：这个设计妙么？妙在哪里？



Linux 中 PCB 的组织方式多种多样，有链表，数组，树结构，哈希表等，这里给出链表结构，链表的头和尾都为 init_task。init_task 是 0 号进程的 PCB，0 号进程就是系统中的 idle(闲逛)进程。系统中的所有进程通过 list_head 结构组成双向循环链表，list_head 是内核中一个独特的双链表结构，请在学堂在线看第一章 1.3《Linux 源代码中的双链表》。

问题：看了 Linux 内核中的双链表，是否颠覆了你对链表的认识，大神们的代码可以写的这么优雅和妙？



下面，我们自己编写一个内核模块，打印系统中所有进程的 PID 和进程名，模块中的代码如下：

这个例子仅仅是打印出 PCB 中的 2 个字段，其实，你可以举一反三，打印出更多字段，来观察进程执行过程中其相关字段的具体值，关于这一内核模块的编写和调试，我们将在动手实践一讲专门讲述。

问题：这个看起来很酷的，居然可以打印进程控制块中的字段，让书上的东西落地了，我能否也可以动手实践呢？可以，请查看视频，动手起来。

因为 Linux 进程控制块中的信息非常多，多大几百个字段，为了有助于对其认识，这里对其给予分类和小结，包括处理器环境信息，状态信息，链接信息，各种标识符，调度信息，进程间通信信息，虚拟内存信息，文件系统信息等等，涉及到进程方方面面，为什么要了解这些信息呢，一旦进程在执行的过程中出问题了，如果你是位老中医，对人体的各个经脉都了解，你就能对症下药。这也是为什么要深入操作系统内部的原因，尽管我们这里介绍的仅仅是冰山一角，但毕竟，你知道一座山到底有多大了。