了下一个要运行的进程是哪一个。

2) 内存映射。指向代码、数据、堆栈段或页表的指针。如果代码段是共享的，代码指针指向共享代码表。当进程不在内存当中时，关于如何在磁盘上找到这些数据的信息也被保存在这里。

3) 信号。掩码显示了哪些信号被忽略、哪些信号需要捕捉、哪些信号被暂时阻塞以及哪些信号在传递当中。

4) 机器寄存器。当内核陷阱发生时，机器寄存器的内容（也包括被使用了的浮点寄存器的内容）会被保存。

5) 系统调用状态。关于当前系统调用的信息，包括参数和返回值。

6) 文件描述符表。当一个与文件描述符有关的系统调用被调用的时候，文件描述符作为索引在文件描述符表中定位相关文件的i节点数据结构。

7) 统计。指向记录用户、进程占用系统CPU时间的表的指针。一些系统还保存一个进程最多可以占用CPU的时间、进程可以拥有的最大堆栈空间、进程可以消耗的页面数等。

8) 内核堆栈。进程的内核部分可以使用的固定堆栈。

9) 其他。当前进程状态。如果有的话，包括正在等待的事件、距离警报时钟超时的时间、PID、父进程的PID以及其他用户标识符、组标识符等。

记住这些信息，现在可以很容易地解释在Linux系统中是如何创建进程的。实际上，创建一个新进程的过程非常简单。为子进程创建一个新的进程描述符和用户空间，然后从父进程复制大量的内容。这个子进程被赋予一个PID，并建立它的内存映射，同时它也被赋予了访问属于父进程文件的权利。然后，它的寄存器内容被初始化并准备运行。

当系统调用fork执行的时候，调用fork函数的进程陷入内核并且创建一个任务数据结构和其他相关的数据结构，如内核堆栈和thread_info结构。这个结构位于进程堆栈栈底固定偏移量的地方，包含一些进程参数，以及进程描述符的地址。把进程描述符的地址存储在一个固定的地方，使得Linux系统只需要进行很少的有效操作就可以找到一个运行中进程的任务数据结构。

进程描述符的主要内容根据父进程的进程描述符来填充。Linux系统只需要寻找一个可用的PID，更新进程标识符散列表的表项使之指向新的任务数据结构即可。如果散列表发生冲突，相同键值的进程描述符会被组成链表。它会把task_struct结构中的一些分量设置为指向任务数组中相应进程的前一/后一进程的指针。

理论上，现在就应该为子进程分配数据段、堆栈段，并且对父进程的段进行复制，因为fork函数意味着父、子进程之间不共享内存。其中如果代码段是只读的，可以复制也可以共享。然后，子进程就可以运行了。

但是，复制内存的代价相当昂贵，所以现代Linux系统都使用了欺骗的手段。它们赋予子进程属于它的页表，但是这些页表都指向父进程的页面，同时把这些页面标记成只读。当子进程试图向某一页面中写入数据的时候，它会收到写保护的错误。内核发现子进程的写入行为之后，会为子进程分配一个该页面的新副本，并将这个副本标记为可读、可写。通过这种方式，使得只有需要写入数据的页面才会被复制。这种机制叫做写时复制。它所带来的额外好处是，不需要在内存中维护同一个程序的两个副本，从而节省了RAM。

子进程开始运行之后，运行代码（shell的副本）调用系统调用exec，将命令名作为exec函数的参数。内核找到并核实相应的可执行文件，把参数和环境变量复制到内核，释放旧的地址空间和页表。

现在必须建立并填充新的地址空间。如果你使用的系统像Linux系统或其他基于UNIX的系统一样支持映射文件，新的页表会被创建，并指出所需的页面不在内存中，除非用到的页面是堆栈页，但是所需的地址空间在磁盘的可执行文件中都有备份。当新进程开始运行的时候，它会立刻收到一个缺页中断，这会使得第一个含有代码的页面从可执行文件调入内存。通过这种方式，不需要预先加载任何东西，所以程序可以快速地开始运行，只有在所需页面不在内存中时才会发生页面错误（这种情况是第3章中讨论的最纯粹的按需分页机制）。最后，参数和环境变量被复制到新的堆栈中，信号被重置，寄存器被全部清零。从这里开始，新的命令就可以运行了。