第7章 进 程

上一章跟踪了操作系统被引导后的启动过程，其中介绍了一些进程概念的重要特性。本章的目的是回过头去透彻地再揭示这些特性。

提出一种广为接受的“进程”定义是极其困难的。这与请哲学家回答“什么是生命 ？”这样的难题类似。如果我们不是钻在某些牛角尖里，那么面临的其他问题就比较容易解决。

前面已经给出了进程定义，“正在执行的一道程序”，这在相当范围内是一个不错的定义。但是，对于＃ 0进程的整个生命期和＃ 1进程的开始阶段，该定义却并不合适。系统中的所有其他进程则清清楚楚地与程序文件中的某一个或几个相关。

可以分两个层次对操作系统进程进行讨论。

在较高的层次上，“进程”是一个重要的组织概念，用其说明一个计算机系统作为一个整体的活动。将计算机系统视作若干进程的组合活动是适当的，每一个进程与一道特定的程序相结合，例如“ shell”或者“编辑程序”。里奇和汤姆森的论文“ The UNIX Time-sharing System”的后半部分就在此层次上对UNIX进行了讨论。

在这一层次上，进程本身被视作系统中的活动实体，而真正的活动部件本体，即处理机和外部设备则被消隐，不引起人们的注意。进程诞生、生长，然后死亡；它们存在的数量在不断变化；它们可以获得并释放资源；它们可以交互作用、合作、冲突、共享资源等等。

在较低的层次上，进程是不活动的实体，它们依靠活动实体，例如处理机才起作用。借助于频繁地使处理机从一个进程映像的执行切换到另一个，就可以产生一种印象：每一个进程映像都连续发生变化，这就导致较高层次上的解释。

我们现在所关心的是较低层次上的解释：进程映像 (process image)的结构、执行的细节以及在进程之间切换处理机的方法。

在UNIX环境中，关于进程可以观察到下列各点：

1) 在“ proc”数组中的一个非空结构意味着一个进程的存在，非空“ proc”结构指的是其元素“ p_stat”非空。

2) 每个进程都有一个“每个进程数据区”(ppda)，其中包含“ user”结构的一个副本。

3) 处理机在其生命期中不是执行此进程就是执行其他进程 (除在两条指令之间的停顿外)。

4) 一个进程可创建或破坏另一个进程。

5) 一个进程可获得并占用各种资源。

7.1 进程映像

里奇和汤姆森在他们的论文中定义一个“进程”为一“映像”的执行，其中“映像”是伪-计算机(pseudo-computer)的当前状态，亦即一个抽象数据结构，它存放在内存或磁盘上。

进程映像涉及2或3个物理上不同的存储区：

1) “ proc ”结构，它被包含在常驻内存的“ proc ”数组中，任一时刻对其都可存取。

2) 数据段，它由“ 每个进程数据区 ”、用户程序数据、(可能的)程序正文和栈组成。

3) 正文段，它并不总是存在，如果存在则由仅包含纯程序正文的段组成，亦即由可再入代码和常数数据组成。

很多程序没有单独的正文段。若有单独的正文段，那么很多执行同一程序的进程就可共享一个副本。

7.2 proc结构(0358)

proc结构常驻内存，它包含15个元素，其中8个是字符型，6个是整型，1个是整型指针。每个元素都提供任何时刻都能存取的信息，特别当进程映像的主要部分已被换出到磁盘上时：

- “ p_stat ”可取7个值中的1个，这7个值定义了7种互斥的状态。请参见0381~0387行。
- “ p_flag ”是6个1位标志的混合物，这6个标志位可单独设置。请参见0391~0396行。
- “ p_addr ”是数据段的地址：
  - 若该数据段在内存中，则这是一个块号。
  - 若该数据段已被换出至磁盘上，则这是一磁盘记录号。
- “ p_size ”是数据段的长度，单位是块。
- “ p_pri ”是当前的该进程优先级。经常对其重新计算，计算式是“ p_nice ”、“ p_cpu ”和“ p_time ”的函数。
- “ p_pid ”、“ p_ppid ”是数值，它们唯一地标识一个进程及其父进程。
- “ p_sig ”、“ p_uid ”、“ p_ttyp ”涉及与外部的通信，亦即来自进程正常域外的消息或“ 信号 ”。
- “ p_wchan ”为“ 睡眠 ”进程(p_stat或者为SSLEEP，或者为SWAIT)，标识其睡眠的原因。
- “ p_textp ”或者为null，或者是指向“ text ”数组(4306)中一项的指针，该text数组项包含了与该正文段有关的重要统计信息。

7.3 user结构(0413)

“ user ”结构的一个副本是每个“ ppda ”的重要组成部份。在任一时刻，只有一个“ user ”副本是可存取的。该副本的名字是“ u ”，总是位于核心态地址0140000，亦即核心态地址空间第7页的起始地址。

“ user ”结构的很多元素在此处还不便于介绍，暂时也不是很有用。源代码上的注释大致说明了“ user ”中每个元素的作用。

此刻，应当注意：

1) “ u_rsav ”、“ u_qsav ”和“ u_ssav ”皆为两个字的数组，被用来存放r5、r6的值。

2) “ u_procp ”指向在“ proc ”数组中相应的“ proc ”结构。

3) “ u_uisa〔16〕”、u_uisd〔16〕”存放页地址和说明寄存器的原型。

4)“u_tsize”、“u_dsize”、“u_ssize”分别是正文段、数据段和栈段的长度，单位是32字块。

余下的元素与下列内容有关：

- 保存浮点寄存器(并非针对PDP11/40)。
- 用户标识。
- 输入/输出操作的参数。
- 文件存取控制。
- 系统调用参数。
- 帐号信息。

7.4 每个进程数据区

“每个进程数据区”对应于核心地址空间第7页的有效部分(低地址部分)。其长度是1024字节。较低的289个字节由“user”结构的一个实例占用，余下的367字用作为核心态栈区(显然，有多少进程就有多少核心态栈。)

当处理机处于核心态时，环境和栈指针r5和r6应指向下列地址范围：

0140441～01437777

超过上界将以段违例自陷，但是下界仅由软件设计的周密考虑而保证不会越出。(应注意的是：UNIX没有使用硬件栈限选项。)

7.5 段

数据段被分配占用单一物理存储区，但是它包含3个不同的部分：

1) 一个“每个进程数据区”。

2) 用于用户程序的数据区。这可进一步划分成程序正文区、初始化数据区和不赋初值数据区。

3) 用于用户程序的栈。

其中，1)的长度总是“USIZE”块。2)和3)的长度由“u.u_dsize”和“u.u_ssize”给出(单位：块)。在进程生存期中后两部分的长度可能更改。

对仅包含纯正文的单独正文段分配一单个物理存储区。该段的内部结构在此处并不重要。

7.6 映像的执行

第7核心态段地址寄存器的设置决定了当前正被执行的映像，因此也就决定了当前进程。若#i进程是当前进程，则该地址寄存器的值为“proc〔i〕.p_addr”。

经常希望将正在核心态执行的进程与正在用户态执行的同一进程两者区分开来。我们将使用术语“核心态进程#i”和“用户态进程#i”分别表示“正在核心态执行的进程#i”和“正在用户态执行的进程#i”。

如果我们选择使进程与特定执行栈相关连，而不是与“ proc ”数组中一项相关连，那么我们就是把核心态进程＃ i和用户态进程＃ i视为单独的进程，而不是单个进程＃ i的两个不同方面。

7.7 核心态执行

第7核心态段地址寄存器的内容必须随当前运行进程的改变而改变。而其他核心态段寄存器在系统启动后就不会再改变。正如前面已经指出的，前 6个核心态页映照至前 6个物理存储器页，同时，第 8个核心态页映照至物理存储器的最高页。第 7段的地址虽然经常变化，但其长度始终相同。

在核心态下运行时，用户态段寄存器的具体设置通常是并无关系的。但是在正常情况下它们总按用户进程正确设置。

环境和栈指针指向在第 7页中的核心态栈区，该栈位于“ user ”结构之上。

7.8 用户态执行

每次在激活一用户进程时，在其前、后总是伴随着相应核心态进程的一次激活。与此相对应，无论何时，只要一进程映像正在用户态执行，那么与其相对应的用户态和核心态寄存器就会被正常设置。

环境和栈指针指向用户态栈区。该栈从第 8用户页的高地址处开始，但可向下扩充，例如占用整个第8页，第7页的一部分或全部等等。

核心态段寄存器的设置相当简单，而用户态段寄存器的设置则相当繁琐。

7.9 一个实例

考虑在PDP11/40上运行的一道程序，其正文部分为1.7页，数据部分为3.3页，栈区为0.7页。(在本例中为方便起见使用了小数，与实际情况稍有差别。)在虚地址空间中的安排示于图7-1中：

在虚地址空间中必须为正文段分配 2整页，而物理存储区则只需1.7页(见图7-2)。

数据和栈区分别要求使用虚地址空间中的 4页和1页，而在物理地址空间中则分别要求使用 3.3页和 0.7页。

整个数据段要求使用四又八分之一页物理存储器。额外的八分之一页用于“ 每个进程数据区 ”，它对应于第7核心态地址页(见图7-3)。

注意数据段各部分的顺序，在各部分之间没有未使用区。

虚地址空间

图 7-1

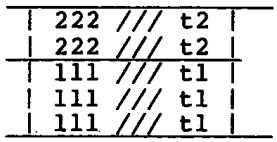

图 7-2

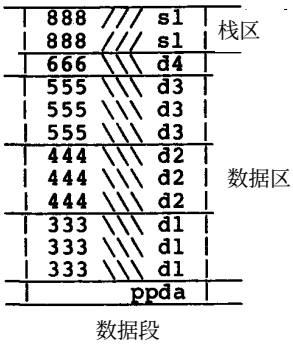

图 7-3

需设置用户态段寄存器以反映下表中的值，其中，“ t ”和“ d ”分别表示正文和数据段的起始地址(单位：块)：

页	地　址	长　度	注　释
1	t+0	1.0	只读
2	t+128	0.7	只读
3	d+16	1.0	
4	d+144	1.0	
5	d+272	1.0	
6	d+400	0.3	
7	?	0.0	未用
8	d+400	0.7	向下扩展

将“ t ”和“ d ”设置为0，则获得存放在数组“ u.u_uisa ”

estabur ”设置的，当一道程序第一次执行，以及存储分
此原型存放在数组“ u.u_uisa ”和“ u.u_uisd ”中。
，则调用“ sureg ”，它将原型数据按需进行适当处理后送
妾复制，但地址寄存器值则必须作调整以反映物理存储区

R证所要求的正文、数据和栈的长度都是合理的。
三文区 (“ i ”空间)和数据区 (“ d ”空间)的分别映照。

上为 0的段地址。“ ap ”和“ dp ”分别指向原型段地址和说

空间，后8个则针对“ d ”空间。

1667：“ nt ”测量正文段所需的32字块数。若nt为非0，则需为正文段分配1页或多页。

若分配多于1页，则除最后1页外，其他各页长度皆为128块(4096字)，而且是只读，它们的相对地址从0开始，各页顺序增加128。

1672：若正文段尚余留部分小于1页，则为其分配下一页的所需部分，并设置相应段地址和说明寄存器。

1677：如分开使用“ i ”和“ d ”空间，则将余下“ i ”页的段寄存器标记为null。

1682：因为所有余下的地址引用数据区 (非正文区)并且相对于此区的起点，所以使“ a ”复位，而此区开始的“ USIZE ”块保留用于“ 每个进程数据区 ”。

1703：从地址空间顶向低地址(向下)分配栈区。

1711：若需为栈区分配页的一部分，则分配该页的高地址部分。 (对于向上扩展的正文和数据区，则分配页的低地址部分。)这要求在说明寄存器中设置一额外位，即“ ED ”位(向下扩展)。

1714：若不使用分开的“ i ”和“ d ”空间，那么到此点为止，只对 16个原型寄存器对中的前8个赋予初值。在这种情况下，后8个从前8个中复制。

7.12 sureg(1739)

“ estabur ”(1724)、“ swtch ”(2229)和“ expand ”(2295)调用此例程以便将原型段寄存器值复制到实际的硬件段寄存器。

1743：从“ proc ”结构的适宜元素取得数据区的基地址。

1744：各原型地址寄存器(对于PDP11/40，有8个原型地址寄存器)的值加上“ a ”，然后存放到相应硬件段地址寄存器。

1752：测试是否已分配分开的正文区，若是则将“ a ”设置为正文区对数据区的相对地址。(注意，此值可能是负值!地址的单位是32字块)

1754：此段代码的样式类似于本例程的开始部分，例外是……

1762：这是一段其作用不易看清楚的代码，它对非可写，亦即正文段的地址寄存器的值进行调整。

“ estabur ”和“ sureg ”代码带有经多次修改的痕迹，它们不像我们所希望的那样精致巧妙。

7.13 newproc(1826)

现在到时候了，让我们仔细观察一下创建新进程的过程，创建的新进程几乎是它们创建者的精确复制器。

1841：“ mpid ”是一个整型值，它在0至32767范围内逐一递增。创建一进程时，要在步进过程中找到一个“ mpid ”的新值，从而为该新进程提供了一个区别于所有其他现存进程的整型数。在寻找“ mpid ”新值时可能进入循环，于是步进得到的值或许会与现存正在使用的值重复，因此要不断进行测试。

1846：在“ proc ”数组中搜索以找到一个空“ proc ”结构(若p_stat的值为null，则该

proc结构为空)。

1860：到达此点时，新找到“ proc ”项的地址存放在“ p ”和“ rpp ”中，而当前进程“ proc ”项的地址则存放在“ up ”和“ rip ”中。

1861：将新进程的属性存放到它的“ proc ”结构中。其中很多是从当前进程处复制的。

1876：新进程继承其父进程的打开文件。对于每一个打开文件增加其引用计数。

1879：如果正文段是共享纯正文段，也就是与数据段分开，则增加相关的引用计数。注意，在此处暂时借用了“ rip ”和“ rpp ”。

1883：增加父进程的当前目录的引用计数。

1889：将环境和栈指针的当前值保存到“ u.u_rsav ”中。“ savu ”是定义在0725行处的汇编语言例程。

1890：恢复“ rip ”和“ rpp ”的值。暂时更改“ u.u_procp ”的值，使其从对应于当前进程的值更改为对应于新进程的值。

1896：试着在内存中找到一个区域，以便在此区域中创建新数据段。

1902：如果在内存中没有一个合适的区域，则应当在磁盘上构造新副本。由于在 1891行引入的不一致性，应当仔细地分析下一段代码。

```
u.u_procp－＞p_addr!=*ka6
```

1903：将当前进程标记为“ SIDL ”，以阻止“ sched ”将其映像换出到磁盘上。

1904：设置

```
rpp－＞p_addr=*ka6;
```

使新进程“ proc ”项协调一致。

[保存至“ u.u_ssav ”。

将数据段复制到磁盘交换区。因为第二个参数为 0，所以

]”。

态。

以保存新“ proc ”项的地址，并一次一块地复制数据段。

数据区”恢复至它的以前状态。

产生有益并变化的进程集。将在第 12章中讨论的“ exec ”

程改变自己的角色从而获得新生。