第6章 系统初启

当UNIX重新启动，也就是在一台空闲机器上装入和初启时将发生一系列事件，本章的目的就是考虑这一系列事件的序列。

对初启过程本身的研究是非常有趣的，但更重要的是它能将系统的很多重要特征以有序的方式展现出来。

在系统崩溃之后可能需要重新启动操作系统。由于某些非常普通、操作上的原因，例如过夜之前的关机，也要频繁地重新启动操作系统。如果我们考虑后一种情况，那么可以认为所有磁盘文件都是完好无损的，没有什么特殊情况需要识别和处理。

我们尤其可以设想在根目录中有一个称为“ /unix ”的文件，它是UNIX操作系统的目标代码。

该文件起始于一组我们正在研究学习的源文件。将这些源文件分别编译，然后连接到一起构成了一个单一目标文件，然后存放在根目录下的 unix文件中。

6.1 操作员的动作

重新启动要求操作员在处理机控制台上执行下列操作：

- 将“ enable/halt ”(启动/停机)开关设置到“ halt ”位置，这样就停止了处理机。
- 将硬件引导装入程序的地址设置到开关寄存器。
- 按下，然后释放“ 装入地址 ”开关。
- 将“ enable/halt ”开关设置到“ enable ”位置。
- 按下，然后释放“ start ”(启动)开关。这样就激活了常驻在处理机中 ROM里的引导程序。

引导装入程序装入一更大的装入程序 (从系统盘的＃0块)，该程序查找并将称为“ /unix ”的文件装入到内存的低地址部分。然后，它将控制转移到已装入＃0地址的指令。

地址0单元中是一条转移指令 (0508行)，它转移到000040单元，其中包含一条跳转 (jump)指令(0522行)，它跳转到标号为“ start ”的指令，该指令在文件“ m40.s ”中(0612行)。

6.2 start(0612)

0613：测试存储管理状态寄存器的“ 启动 ”位 SR0。若该位设置，则处理机将一直执行两条指令的循环。在启动系统之前，当操作员触发在控制台上的“ 清除 ” (“ clear ”)按钮时，该寄存器通常被清除。

对于这一循环的必要性已提出了很多理由。最主要的一条是：在双总线超时错情况下，处理机将转移到＃0单元，在这种情况下不应当允许程序向前执行。

0615：“ reset ”(清除)清除位，启动所有外部设备控制和状态寄存器。

现在，系统将运行在核心态方式，而存储管理部件则未启动。

0619：KISA0和KISD0处于物理存储空间的高地址部分，它们是第一对核心态段寄存器的地址。前6个核心态说明寄存器被赋予的初值是 077406，它说明该段长度为 4K字，存取控制是读/写。

前6个核心态地址寄存器被赋予的初值分别是 0、0200、0400、0600、01000和01200。

对前6个核心态段说明和地址寄存器赋初值的结果是：它们分别指向了物理存储器的前 6个4K字段。于是，对于在 0137777范围内的核心态地址，将它们翻译至物理地址是轻而易举的。

0632：“ _end ”是装入程序的一个伪变量，它定义程序代码和数据区的范围。此值被归整到下一个64字节的整倍数并存放在第7个段(段＃6)地址寄存器中。

注意：此寄存器的地址存放在“ ka6 ”中，所以此寄存器的内容可以用“ * ka6 ”存取。

0634：第7段说明寄存器装入的值表示：该段长度是 16*32=512字，其存取控制字段是读/写。

将8进制017左移8个位置，然后与6相“ 或 ”则得值007406。

0641：第8段映照为物理地址空间的最高 4K字段。

应当注意，在存储管理并未启动情况下，此种释译已经实施，亦即 32K程序地址空间中最高4K字段的地址自动映照为物理地址空间中的最高 4K字段。

我们可以注意到，从此点开始核心态段寄存器除一个例外都将保持不变，该例外是第 7个核心态段地址寄存器。

UNIX对该寄存器进行显式操作使其指向物理存储器中经常变动的一个位置。每一个这种位置都是一个 512字长区间的起始点，该区间称之为“ 每个进程数据区 ” (per process data area)。

现在第7核心态地址寄存器被设置为指向一个段，该段将成为＃ 0 进程的 ppda。

0646：将栈指针设置为指向ppda的最高字。

0647：使SR0值从0增为1，这样就方便地设置了“ 存储管理启动 ”位。

从此开始，所有程序地址都由存储管理硬件释译为物理地址。

0649：“ bss ”指的是程序数据区的第二部分，装入程序对该部分并不赋初值 (参见UPM中的“ A.OUT(V) ”)。此部分的低、高地址分别由装入程序的伪变量“ _ edata ”和“ _end ”定义。

0668：更改处理机状态字(PS)以指明“ 前状态 ”是“ 用户态 ”。这为检查和初始化非核心态地址空间外的物理存储器作好了准备。 (这涉及到使用专用指令“ mtpi ”和“ mfpi ”(至/从前指令空间作移动，以及对用户态段寄存器的处理。)

0669：调用“ main ”过程(1550)。

后面将观察到：“ main ”调用“ sched ”，而“ sched ”则不会终止。那么，“ start ”的最后3条指令(0670、0671和0672行)是否需要，又有什么用处呢?这有些令人迷惑。对此问题的答案将在后面说明。读者可以先推敲一下“ 为什么 ”，“ 这些行究竟是做什么的？”。

6.3 main(1550)

在进入此过程时：

1) 处理机在优先级0、核心态运行，而前状态为用户态。

2) 核心态段寄存器已被设置、存储管理单元已启动。

3) 操作系统使用的所有数据区已初始化。

4) 栈指针(SP或r6)指向一个字，其中包含返回至“ start ”的地址。

1559：因为“ start ”执行的初始化中已将“ updlock ”设置为0，所以“ main ”的第一个动作似乎是多余的。

1560：“ i ”被赋初值为＃０进程ppda区后第一个32字块块号。

1562：第一对用户态段寄存器用作在物理内存较高区域的“ 移动窗口 ” (moving window)。

在该窗口的每一个位置，都试图读该窗口中的第 1个可存取的字(用fuibyte)。如果这一读操作失败，则表示已到达物理存储器的尾端。否则将窗口中的 32个字都赋初值为 0(用clearseg(0676))，并将该块加至可用存储器列表中，然后将窗口前移 32个字。

“ fuibyte ”和“ clearseg ”都位于“ m40.s ”中。在正常情况下，“ fuibyte ”将返回0~223之间的一个正值。但是，在异常情况下，亦即对该存储单元的访问没有得到响应时，“ fuibyte ”将返回－1。(产生这种结果的方式还是有点含糊不情，在第10章中将对此作解释。)

1582：“ maxmem ”规定了可供用户程序使用的主存最大总量。它是下列两个值中的较小值：

- 物理上可用的存储器(maxmem)。
- 一个装置可定义的参数(MAXMEM)(0135)。

最后的限制取决于PDP11系统结构。

1583：“ swapmap ”定义了在交换磁盘(swapping disk)上的可用空间，当用户程序被换出主存时，可使用该磁盘空间。初始化后这是一个连续的单一区域，其长度是“ nswap ”，起始相对地址是“ swplo ”。注意，“ nswap ”和“ swplo ”在“ conf.c ”(4697行和4698行)中初始化。

1589：很快就会对此行及下4行的作用进行讨论。

1599：UNIX设计假定系统中有一个时钟，它以线频(亦即50Hz或60Hz)中断处理机。

有两种可供使用的时钟类型：一种是线频时钟 (KW11-L)，其控制寄存器在单总线上的地址是777546；另一种是可编程实时时钟(KW11-P)，其地址是777540(1509行、1510行)。

UNIX并不假定系统中存在哪一种时钟。首先它试图读线频时钟的状态字。若成功，则初启该时钟，而另一个(若存在)则弃而不用。若不成功，则试探另一个时钟。若对两者的检测都没有成功，则调用“ panic ”，它等效于停止系统，并向操作员传送一条出错消息。

先将一对用户态段寄存器设置为适当值 (1599，1600行)，然后用“ fuiword ”引用时钟状态字，若产生总线超时错，则表示不存在该时钟。

1607：时钟用下列语句初始化

```
*lks=0115;
```

此操作的结果是：经20ms后该时钟将向处理机提出中断请求。该中断可在任何时刻发生，但是为了讨论的方便，可以假定在完成系统启动之前，不会发生中断。

1613：“ cinit ”(8234)初始化字符缓存池。参见第23章。

1614：“ binit ”(5055)初始化大缓存池(亦即块缓存池)。参见第17章。

1615：“ iinit ”(6922)为根设备(root device)初始化装配表的表目项。参见第20章。

6.4 进程

“ 进程 ”是一个已多次出现的术语。目前与我们目的相适应的进程定义可以是：正在执行的程序。

UNIX中进程表示的细节将在下一章中讨论。现在我们想提请读者注意的是：每个进程都包含一取自“ proc ”数组的“ proc ”结构和一个“ 每个进程数据区 ” (ppda)，它包括一“ u ”结构的拷贝。

6.5 proc〔0〕的初始化

从1589行开始对“ proc〔0〕”进行初始化。proc〔0〕中各元素在先前已被赋初值0，现在只改变其中4个元素的值：

1)“ p_stat ”被设置为“ SRUN ”，这意味着＃0进程已为运行准备就绪。

2)“ p_flag ”被设置为“ SLOAD ”和“ SSYS ”。前者意味着该进程在内存中(没有将它换出到磁盘上)，后者意味着决不应将此进程换出到磁盘上。

3)“ p_size ”被设置为“ USIZE ”。

4)“ p_addr ”被设置为核心态段地址寄存器＃6的内容。

将会观察到＃0进程已获得长度为“ USIZE ”(单位：块)的区(就是ppda的长度)，其起点紧跟操作系统数据区的结尾处(_end)。

该区第一块的序号数存入“ p_addr ”中以供后用。“ start ”(0611)中曾将该区清为0，现包含称为“ u ”的“ user ”结构的一个副本。

在1593行，proc〔0〕”的地址存入“ u.u_procp ”，亦即“ proc ”和“ u ”结构相互关连。

1627：下一章将详细讨论“ newproc ”(1826)。

简而言之，“ newproc ”初始化第2个“ proc ”结构(proc〔1〕)，并在内存中分配第2个“ ppda ”。这里＃0进程“ ppda ”的一个副本，其区别只有1项，第2个“ ppda ”中的“ u.u-procp ”的值是&proc〔1〕。

这里应当注意的是：1889行中调用了“ savu ”(0725)，在构造第2个“ ppda ”副本之前，它将环境和栈指针的当前值保存在“ u.u-rsav ”中。

从1918行我们可以看到，“ newproc ”的返回值是0，所以将不会执行1628行至1635行的语句。

1637：调用“ sched ”(1940)，可以先大致观察一下该过程，它包含一无限循环，所以决

不会返回!

6.6 sched(1940)

在此阶段我们所关心的只是，当第一次进入“ sched ”时会发生些什么。

1958：“ spl6 ”是一个汇编例程(1292)，它将处理机优先级设置为6(请对照“ m40.s ”中的“ spl0 ”、“ spl4 ”、“ spl5 ”和“ spl7 ”)。

当处理机处于优先级6级时，只有优先级为7级的设备才能中断它。从此开始到调用“ spl0 ”(1976行)之间，优先级为6级的时钟就被禁止对处理机的中断。

1960：搜索“ proc ”数组，寻找其状态为SRUN并且未“ 装入 ”内存的进程。

＃0和＃1进程的状态为“ SRUN ”并且已“ 装入 ”内存，其余所有进程的状态为0，它等效于“ 未定义 ”或“ NULL ”。

1966：该搜索失败(n仍为－1)。使“ runout ”标志值非负，表明没有进程，它们准备好运行并且“ 已换出 ”到磁盘上。

1968：调用“ sleep ”(等待此种事件发生)，调用时提供的优先权参数为“ PSWP ”(==100)，该优先权在此进程被唤醒后起作用，“ PSWP ”处在“ 非常紧急 ”的优先类中。

6.7 sleep(2066)

2070：“ PS ”是处理机状态字的地址。处理机状态存放在寄存器“ S ”中(0164，0175)。

2071：“ rp ”被设置为当前进程在“ proc ”数组中表目项的地址，也就是使“ rp ”指向当前进程的proc结构(此时，当前进程仍旧是proc〔0〕)。

2072：“ pri ”为负，所以转移至“ else ”部分，将当前进程(＃0)状态设置为“ SSLEEP ”。进入睡眠状态的原因以及唤醒后的优先权皆记入该进程的“ proc ”结构(proc〔0〕)中。

2093：调用“ swtch ”。

6.8 swtch(2178)

2184：“ p ”是一个静态变量(2180)，这意味着其值被初始化为0(1566)，而且在两次调用之间其值保持不变。在第一次调用“ swtch ”时，将“ p ”设置为指向“ proc〔0〕”。

2189：调用“ savu ”，它将当前进程的栈指针和环境指针保存到“ u.u_rsav ”中。

2193：调用“ retu ”：

1) 按传递过来的参数值设置核心态＃6段地址寄存器(这造成当前进程的更改)。

2) 按更改过的当前进程设置栈和环境指针，该当前进程即将恢复执行。

此处对“ savu ”和“ retu ”的顺次调用的组合构成了“ 合作例程跳转(coroutine jump)(请对照Cyber系统的“ 交换跳转 ”(exchauge jump)，IBM/360机上的“ 装入PSW ”或B6700上的“ 移动栈 ”(Move Stack)。

2201：搜索进程集以找到一进程，其状态是“ SRUN ”并且已装入内存，而且在所有这些进程中其“ p_pri ”值最小。(进程p_pri值小，则其优先权高。)

该搜索成功并且找到的是＃１进程。(＃０进程的状态刚从SRUN变为SSLEEP，所以它不再满足搜索寻找条件)。

2218：因为“ p ”非空，所以不进入空闲(idle)循环。

2228：“ retu ”(0740)造成合作例程跳转至＃１进程，它成为当前进程。

＃1进程是什么呢?它基本上是＃０进程的复制品。

在调用“ retu ”之前并未调用“ savu ”，事实上在此之前已经保存了所需的信息。(在什么地方?)

2229：“ sureg ”是一例程(1738)，它将相应于当前进程的值复制到用户态寄存器中。这些值在早先已存入“ u.u_uisa ”和“ u.u_uisd ”数组。

最早一次“ sureg ”调用复制0，它并无实际作用。

2240：“ SSWAP ”标志没有设置，所以现在可以忽略这一段令人迷惑的程序 (2239)。

2247：最后，“ swtch ”返回值１。但是该“ return ”究竟返回到何处呢 ?不是返回到“ sleep ”!(Not to sleep!)

在“ return ”之前用栈指针和环境指针进行一组通用寄存器值的设置。这些值恰好(在返回之前)等于最近一次“ savu(u.u_rsav) ”执行时的值。

现在＃1进程刚刚启动，它虽然从未执行过“ savu ”，但在复制＃０进程之前(由main调用newproc实现)，相应值已存放到了“ u.u_rsav ”中。

于是，在这种情况下，从“ swtch ”就返回到“ main ”，返回值为1。(对此请再查看一次，保证你对此已理解了!)

———.

0进程创造了一个它的复制品＃1进程，＃0进程则进入睡
进程，并返回到“ main ”，其返回值是1。现在让我们继续

条语句，它是一条if语句，其条件表达式是“ newproc( ) ”。
＃１进程找到一块新的较大区 (从USIZE∗32字扩大主
据区复制到其中。
区仅由“ ppda ”组成，没有数据和栈区。现在释放原先

原型”段寄存器组的值，它们被存放在“ u.u_uisa ”和
使用。作为其最后的动作，“ estabur ”调用“ sureg ”。
别是正文、数据和栈区的长度，以及一个指示标志，用其
同的地址空间。(在PDP11/40机上，这两个区并不分别存
32字。
汇编语言例程，它将核心空间中一指定长度的数值复制到
复制到用户空间从地址0开始的一个区域中。

1635：此“ return ”并无特殊之处。它从“ main ”返回至“ start ”(0670)，其中最后3条指令的效果是：在用户态执行用户态地址空间中位于＃ 0地址的指令，也就是执行“ icode ”中第一条指令的副本。随后执行的指令也是“ icode ”中指令的副本。

到达这一点时，系统初始化全部完成。

1＃进程正在运行，在全部意义上它都是一个正常的进程。它的初始形式是下列“ C ”程序经编译、装入和执行后形成的：

```
char *init "/etc/init";
main ( ) {
execl (init, init, 0);
while (1);
}
```

其等效的汇编语言程序是：

```
        sys exec
        init
        initp
        br
initp:  init
        0
init:   </etc/init\0>
```

如果系统调用“ exec ”失败(例如不能找到文件/etc/init)，则该进程进入一无限循环，除非发生时钟中断，否则处理机就停止不前。

“ etc/init ”所执行的功能在UPM的“ INIT(Ⅷ)中说明。