第23章 字符处理

对字符特殊设备提供的缓存是一组 4个字的块，每一块可存放 6个字符。该存储块的原型是“ cblock ”(8140)，其中包含一指向cblock结构的指针，另外一部分是字符数组，其中可存放6个字符。

另一个重要的数据结构类型为“ clist ”(7908)，其中包含一字符计数器以及头、尾指针，这种结构用作“ cblock ”类型块列表的表头。

当前未用的各“ cblcck ”通过它们的头指针连接成一个空闲“ cblock ”列表，该列表表头指针是“ cfreelist ”(8149)。列表中最后一个元素的头指针值为“ NULL ”。

一个“ cblock ”列表为一字符列表提供了存储区。“ putc ”过程的作用是将一个字符加至这种列表的尾部，“ getc ”则从列表首取出一个字符。

图23-1～图23-4用实例说明了对这种列表增、删字符时，该列表的变化情况。

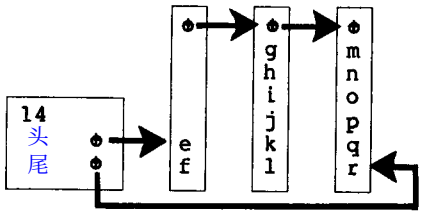

图 23-1

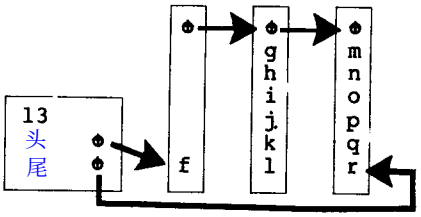

图 23-2

开始时假定该列表中已包含 14个字符，它们是“ efghijk lmnopqr ”。注意，该列表的头、尾指针分别指向第1个字符(在这里是e)和最后1个字符(在这里是r)的位置。若“ getc ”取走第1个字符“ e ”，则图23-1中列表的情况改变成图 23-2。字符计数减1(从14减为13)，头指针则向前移动一个字符位置。

如果再从该列表中取走字符“ f ”，则该列表的状况变为图 23-3。字符计数再减 1，原先的第1“ cblock ”不再包含任何有用信息，已被返回至“ cfreelist ”列表。现在，该列表的头指针指向在第 2个“ cblock ”中的第1个字符(g)。

现在可以提出的一个问题是：当从列表中取出一字符时，如何才能检测出这是第 1种变化情况，还是第2种情况，从而采取不同的合适处理方法泥？

如果读者还没猜想到，那么可以告诉你此问题的答案是：将头指针地址值模除 8，然后检查其结果值。因为在一台二进制计算机上除以 8这种运算是非常容易执行的，因此为什么在每个“ cblock ”中只选择存放6个字符的理由也就十分明显了。

将1个字符加至列表中，使图 23-3变成图23-4。

因为图23-3中的最后一个“ cblock ”是满的，所以从“ cfreelist ”中分配一新“ cblock ”，

并将其连接到“ cblock ”列表中。在该列表中增加 1个字符后，字符计数和尾指针都作了相应调整。

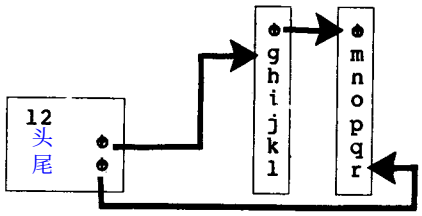

图 23-3

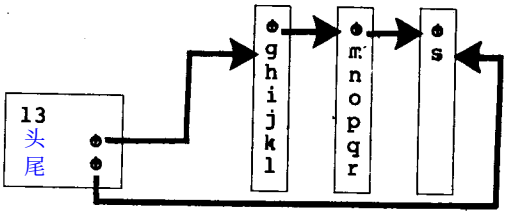

图 23-4

23.1 Cinit(8234)

此过程仅由“ malin ”(1613)调用一次，它将一组字符缓存连接构成“ cfreelist ”指向的空闲列表，然后计字符设备类型数。

8239：“ ccp ”是在“ cfree ”数组(8146)中第1个字的地址。

8240：取大于“ CCP ”的下一个8的整倍数，在“ cfree ”数组中标出取“ cblock ”的边界。注意，从“ cfree ”数组中划分出的最后一个“ cblock ”的起始地址要小于等于“ cfree〔NLIST－1〕”的起始地址，而非“ cfree〔NLIST〕”的起始地址。

8241：使刚取出的“ cblock ”的第1个字指向自由列表的当前头部。注意，“ C_next ”定义在8141行中，而“ cfreelist ”的初始值为“ NULL ”。

8242：更新“ cfreelist ”，使其指向新的列表头部。

看“ cdevsw ”，可见“ nchrde ”将被设置为16，而更合适

```
flushtty (8258, 8259, 8264)
canon    (8292)  pcread  (8688)
ttstart  (8520)  pcstart (8714)
ttread   (8544)  lpstart (8971)
pcclose  (8673)
```

构的地址。

栈上保存原先的处理机状态字和 r2的值。

高于字符设备的中断优先级)。

第 1个字(亦即字符计数)。将该结构第 2个字的值移送给

NULL)，则跳转至0961行。

使r2增1(副作用)。

符号位。

0940：将更新后的头指针存放回“ clist ”结构。(在以后还可能对此作变更。)

0941：使字符计数减1，然后若其值仍为正，则跳转至0947行。

0942：该列表现在为空，所以将头、尾字符指针复位为“ NULL ”。跳转至0952行。

0947：检查r2的最低3位。若这3位值非0，则跳转至0957行(然后在0959行返回至调用例程)。

0949：在此点上， r2指向该“ cblock ”后的下一个字符位置。将存放在该“ cblock ”第1个字(亦即，地址为(r2-8)的单元)的值移送至“ clist ”中的头指针，这样该头指针就指向了列表中原先的第2个“ cblock ”。(注意，r1在0941行作为副作用已经增1。)

0950：此最后存放的值需加2(参看图23-2和图23-3)。

0952：在此点上，由r2决定的“ cblock ”应返回至“ cfreelist ”。无论r2是指在“ cblock ”中或刚刚超过此“ cblock ”。减r2，使其指在“ cblock ”之中。

0953：清r2的最低3位，使其指向该“ cblock ”。

0954：将该“ cblock ”连接至“ cfreelist ”。

0957：从栈中取值恢复r2和PS。

0961：在此点上，因为已检测到头指针为“ NULL ”，所以已知该列表将为空。确保尾指针也是“ NULL ”。

0962：将-1送至r0，作为列表为空时的返回值。

23.3 putc(0967)

此过程由下列过程调用：

```
canon        (8323)
ttyinput     (8355,8358)
ttyoutput    (8414, 8478)
pcrint       (8730)
pcoutput     (8756)
lpoutput     (8990)
```

调用时的两个参数是：一个字符以及一个“ clist ”结构

“ getc ”和“ putc ”的功能，关系密切，这两个过程的

这一理由，所以我们不再详细分析“ putc ”的代码，而将此

应当提请注意的是：若“ putc ”需要一个新的“ cbloc

失败。在此情况下，它返回一非0值(1002行)，而成功执行

注意，此处讨论的“ getc ”和“ putc ”过程与UPM的C

过程并无直接联系。

23.4 字符集

UNIX使用示于UPM的ASCII(V)中的全ASCII字符集。

明，但这并不总是无可非议的，事实上我们在这里就需要对

“ ASCII ”是“ American Standard Code for Information

标准代码)首字母缩略词。

控制字符

在128个ASCII字符中，前32个字符是非图形字符，可用于控制传输或显示的某些方面。UNIX所识别的控制字符是：

数 值	助 记 符	说 明	UNIX名字
004	eot	传输结尾或(Control-D)	004
010	bs	退格	010
011	ht	(水平)制表	′＼t′
012	nl	新行或换行	FDRM
014	np	新页或换页	′＼n′
015	cr	回车	′＼r′
034	fs	文件分隔符或退出	CQUIT
040	sp	空格	′ ′
0177	del	删除	CINTR

应当注意的是其中最后2个字符属于后96个字符，也就是图形字符部分。

23.5 图形字符

有96个图形字符。其中的2个，空格和删除是不可见的，可以将它们归入控制字符类。图形字符可以分成3组，每组32个字符，各组的特性大致是：

1) 数字和特殊字符。

以后2组同样也包括一些特殊字符。特别是，最后 1组包含

，例如行式打印机和终端经常被说成支持 96ASCⅡ字符集

有其他ASCⅡ图形符的设备被说成支持64ASCⅡ字符集。的符号，亦即“～”、“{”、“｜”、“}”、“~”。注意，因为乃可得到支持。

后一组中的设备也可被称之为“大写字母”设备。

有时某些图形符可以是非标准的，例如“←”代替“—”，虽然这通常不是致命的错误，但却可能造成不便。

23.6 UNIX惯例

在日常英语中我们更多使用的是小写字母，UNIX采用了这种人们普遍认可的惯例。来自“大写字母”终端的字母字符在接收时立即从大写字母翻译为小写字母。若在其之前有一个斜线符，那么一个小写字母以后也可翻译为大写字母。对于这种终端的输出，大、小写字母都映照为大写字母。

5个“大写”特殊字符可等效为如下形式：

字 符	行式打印机	终 端
{	↑	\'
|	←	\(
}	+	\!
~	^	\)
		\^

由于下述原因，行式打印机和终端的约定是不同的：

1) 对于行式打印机而言，水平对齐是非常重要的，并无太多困难就可以打印组合、重打字符(在此情况下使用(“－”号)。

2) 对于终端，水平对齐并不被认为非常重要，用退格提供重打字符在很多VDU上并不工作，因为同一图形约定既用于输入也用于输出，所以所用的符号应当尽可能便于打印。

23.7 maptab(8117)

此数组用于翻译从终端输入的字符，这种字符之前有一个反斜线符“\”。

有3个总具有特殊意义的字符，它们是：004(eot)、#和@，当要将它们作为一般字母处理时，应在其前加一个反斜线符。这3个字符在“maptab”中位于它们的自然位置(亦即它们在ASCⅡ表中的位置)。例如#的代码是043，

maptab〔043〕==043。

在“maptab”中的其他非空字符涉及来自“大写字母”设备输入字符的翻译问题，它们不处于它们的自然位置而是在它们的等效字符位置，因为“\(”将被解释为“{”，所以“{”处于“(”的自然位置。

注意关于字母字符的情况。我们要记住的只是：在识别任一反斜线之前，字母字符都先被翻译成小写字母。

23.8 partab(7947)

此数组类似于“maptab”，它由256个字符组成。不幸的是，在UNIX操作系统源代码部分略去了“partab”的初始化部分。这确实是需要的，所以在下面给出：

```
char partab [] {

0001,0201,0201,0001,0201,0001,0001,0201,
0202,0004,0003,0205,0005,0206,0201,0001,
0201,0001,0001,0201,0001,0201,0201,0001,
0001,0201,0201,0001,0201,0001,0001,0201,
0200,0000,0000,0200,0000,0200,0200,0000,
0000,0200,0200,0000,0200,0000,0000,0200,
0000,0200,0200,0000,0200,0000,0000,0200,
0200,0000,0000,0200,0000,0200,0200,0000,
0200,0000,0000,0200,0000,0200,0200,0000,
0000,0200,0200,0000,0200,0000,0000,0200,
0000,0200,0200,0000,0200,0000,0000,0200,
0200,0000,0000,0200,0000,0200,0200,0000,
0000,0200,0200,0000,0200,0000,0000,0200,
0200,0000,0000,0200,0000,0200,0200,0000,
0200,0000,0000,0200,0000,0200,0200,0000,
0000,0200,0200,0000,0200,0000,0000,0201

};
```

“partab”的每一个元素都是一个8位字符，使用适当的位屏蔽(0200和0177)，可将其解释为包含2个部分的结构：

第7位	奇偶位
第3～6位	不使用，总是0
第0～2位	代码号

当计算机传送一字符时，偶校验位被附加至第7位ASCⅡ代码以构成一8位代码。

代码号由“ttyoutput”使用，以便将每个字符归入7类中的1类，操作系统按字符类决定在传送下一个字符之前的延迟时间。(对于机械式打印机，使打印托架从行尾回到行首等操作需常重要的。)