```
/*
 *  linux/kernel/vsprintf.c
 *
 *  (C) 1991  Linus Torvalds
 */

/* vsprintf.c -- Lars Wirzenius & Linus Torvalds. */
/*
 * Wirzenius wrote this portably, Torvalds fucked it up :-)
 */
// Lars Wirzenius 是 Linus 的好友，在 Helsinki 大学时曾同处一间办公室。在 1991 年夏季开发 Linux
// 时，Linus 当时对 C 语言还不是很熟悉，还不会使用可变参数列表函数功能。因此 Lars Wirzenius
// 就为他编写了这段用于内核显示信息的代码。他后来(1998 年)承认在这段代码中有一个 bug，直到
// 1994 年才有人发现，并予以纠正。这个 bug 是在使用*作为输出域宽度时，忘记递增指针跳过这个星
// 号了。在本代码中这个 bug 还仍然存在（130 行）。 他的个人主页是 http://liw.iki.fi/liw/

#include <stdarg.h>        // 标准参数头文件。以宏的形式定义变量参数列表。主要说明了-个
                           // 类型(va_list)和三个宏(va_start, va_arg 和 va_end)，用于
                           // vsprintf、vprintf、vfprintf 函数。
#include <string.h>        // 字符串头文件。主要定义了一些有关字符串操作的嵌入函数。

/* we use this so that we can do without the ctype library */
/* 我们使用下面的定义，这样我们就可以不使用 ctype 库了 */
#define is_digit(c)     ((c) >= '0' && (c) <= '9')   // 判断字符 c 是否为数字字符。

// 该函数将字符数字串转换成整数。输入是数字串指针的指针，返回是结果数值。另外指针将前移。
static int skip_atoi(const char **s)
{
        int i=0;

        while (is_digit(**s))
                i = i*10 + *((*s)++) - '0';
        return i;
}

// 这里定义转换类型的各种符号常数。
#define ZEROPAD 1               /* pad with zero */         /* 填充零 */
#define SIGN    2               /* unsigned/signed long */  /* 无符号/符号长整数 */
#define PLUS    4               /* show plus */             /* 显示加 */
#define SPACE   8               /* space if plus */         /* 如是加，则置空格 */
#define LEFT    16              /* left justified */        /* 左调整 */
#define SPECIAL 32              /* 0x */                    /* 0x */
#define SMALL   64              /* use 'abcdef' instead of 'ABCDEF' */ /* 使用小写字母 */

// 除操作。输入：n 为被除数，base 为除数；结果：n 为商，函数返回值为余数。
// 参见 4.5.3 节有关嵌入汇编的信息。
#define do_div(n,base) ({ \
int __res; \
__asm__("divl %4":"=a" (n),"=d" (__res):"0" (n),"1" (0),"r" (base)); \
__res; })

// 将整数转换为指定进制的字符串。
```

```
   // 输入：num-整数；base-进制；size-字符串长度；precision-数字长度(精度)；type-类型选项。
   // 输出：数字转换成字符串后指向该字符串末端后面的指针。
40 static char * number(char * str, int num, int base, int size, int precision
41         ,int type)
42 {
43         char c,sign,tmp[36];
44         const char *digits="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
45         int i;
46
   // 如果类型 type 指出用小写字母，则定义小写字母集。
   // 如果类型指出要左调整（靠左边界），则屏蔽类型中的填零标志。
   // 如果进制基数小于 2 或大于 36，则退出处理，也即本程序只能处理基数在 2-32 之间的数。
47         if (type&SMALL) digits="0123456789abcdefghijklmnopqrstuvwxyz";
48         if (type&LEFT) type &= ~ZEROPAD;
49         if (base<2 || base>36)
50                 return 0;
   // 如果类型指出要填零，则置字符变量 c='0'，否则 c 等于空格字符。
   // 如果类型指出是带符号数并且数值 num 小于 0，则置符号变量 sign=负号，并使 num 取绝对值。
   // 否则如果类型指出是加号，则置 sign=加号，否则若类型带空格标志则 sign=空格，否则置 0。
51         c = (type & ZEROPAD) ? '0' : ' ';
52         if (type&SIGN && num<0) {
53                 sign='-';
54                 num = -num;
55         } else
56                 sign=(type&PLUS) ? '+' : ((type&SPACE) ? ' ' : 0);
   // 若带符号，则宽度值减 1。若类型指出是特殊转换，则对于十六进制宽度再减少 2 位(用于 0x)，
   // 对于八进制宽度减 1（用于八进制转换结果前放一个零）。
57         if (sign) size--;
58         if (type&SPECIAL)
59                 if (base==16) size -= 2;
60                 else if (base==8) size--;
   // 如果数值 num 为 0，则临时字符串='0'；否则根据给定的基数将数值 num 转换成字符形式。
61         i=0;
62         if (num==0)
63                 tmp[i++]='0';
64         else while (num!=0)
65                 tmp[i++]=digits[do_div(num,base)];
   // 若数值字符个数大于精度值，则精度值扩展为数字个数值。
   // 宽度值 size 减去用于存放数值字符的个数。
66         if (i>precision) precision=i;
67         size -= precision;

   // 从这里真正开始形成所需要的转换结果，并暂时放在字符串 str 中。
   // 若类型中没有填零(ZEROPAD)和左靠齐（左调整）标志，则在 str 中首先
   // 填放剩余宽度值指出的空格数。若需带符号位，则存入符号。
68         if (!(type&(ZEROPAD+LEFT)))
69                 while(size-->0)
70                         *str++ = ' ';
71         if (sign)
72                 *str++ = sign;
   // 若类型指出是特殊转换，则对于八进制转换结果头一位放置一个'0'；而对于十六进制则存放'0x'。
73         if (type&SPECIAL)
74                 if (base==8)
```

```
75                      *str++ = '0';
76              else if (base==16) {
77                      *str++ = '0';
78                      *str++ = digits[33];    // 'X'或'x'
79              }
    // 若类型中没有左调整（左靠齐）标志，则在剩余宽度中存放 c 字符（'0'或空格），见 51 行。
80      if (!(type&LEFT))
81              while(size-->0)
82                      *str++ = c;
    // 此时 i 存有数值 num 的数字个数。若数字个数小于精度值，则 str 中放入（精度值-i）个'0'。
83      while(i<precision--)
84              *str++ = '0';
    // 将数值转换好的数字字符填入 str 中。共 i 个。
85      while(i-->0)
86              *str++ = tmp[i];
    // 若宽度值仍大于零，则表示类型标志中有左靠齐标志。则在剩余宽度中放入空格。
87      while(size-->0)
88              *str++ = ' ';
89      return str;     // 返回转换好的指向字符串末端后的指针。
90 }
91
    // 下面函数是送格式化输出到字符串中。
    // 为了能在内核中使用格式化的输出，Linus 在内核实现了该 C 标准函数。
    // 其中参数 fmt 是格式字符串；args 是个数变化的值；buf 是输出字符串缓冲区。
    // 请参见本代码列表后的有关格式转换字符的介绍。
92 int vsprintf(char *buf, const char *fmt, va_list args)
93 {
94      int len;
95      int i;
96      char * str;             // 用于存放转换过程中的字符串。
97      char *s;
98      int *ip;
99
100     int flags;              /* flags to number() */
101                             /* number()函数使用的标志 */
102     int field_width;        /* width of output field */
                                /* 输出字段宽度*/
103     int precision;          /* min. # of digits for integers; max
104                                number of chars for from string */
                                /* min. 整数数字个数；max. 字符串中字符个数 */
105     int qualifier;          /* 'h', 'l', or 'L' for integer fields */
106                             /* 'h', 'l',或'L'用于整数字段 */
    // 首先将字符指针指向 buf，然后扫描格式字符串，对各个格式转换指示进行相应的处理。
107     for (str=buf ; *fmt ; ++fmt) {
    // 格式转换指示字符串均以'%'开始，这里从 fmt 格式字符串中扫描'%'，寻找格式转换字符串的开始。
    // 不是格式指示的一般字符均被依次存入 str。
108             if (*fmt != '%') {
109                     *str++ = *fmt;
110                     continue;
111             }
112
    // 下面取得格式指示字符串中的标志域，并将标志常量放入 flags 变量中。
113             /* process flags */
```

```
114                 flags = 0;
115                 repeat:
116                         ++fmt;          /* this also skips first '%' */
117                         switch (*fmt) {
118                                 case '-': flags |= LEFT; goto repeat;    // 左靠齐调整。
119                                 case '+': flags |= PLUS; goto repeat;    // 放加号。
120                                 case ' ': flags |= SPACE; goto repeat;   // 放空格。
121                                 case '#': flags |= SPECIAL; goto repeat; // 是特殊转换。
122                                 case '0': flags |= ZEROPAD; goto repeat;  // 要填零(即'0')。
123                                 }
124
    // 取当前参数字段宽度域值，放入 field_width 变量中。如果宽度域中是数值则直接取其为宽度值。
    // 如果宽度域中是字符'*'，表示下一个参数指定宽度。因此调用 va_arg 取宽度值。若此时宽度值
    // 小于 0，则该负数表示其带有标志域'-'标志（左靠齐），因此还需在标志变量中添入该标志，并
    // 将字段宽度值取为其绝对值。
125                 /* get field width */
126                 field_width = -1;
127                 if (is_digit(*fmt))
128                         field_width = skip_atoi(&fmt);
129                 else if (*fmt == '*') {
130                         /* it's the next argument */     // 这里有个 bug，应插入++fmt;
131                         field_width = va_arg(args, int);
132                         if (field_width < 0) {
133                                 field_width = -field_width;
134                                 flags |= LEFT;
135                         }
136                 }
137
    // 下面这段代码，取格式转换串的精度域，并放入 precision 变量中。精度域开始的标志是'.'。
    // 其处理过程与上面宽度域的类似。如果精度域中是数值则直接取其为精度值。如果精度域中是
    // 字符'*'，表示下一个参数指定精度。因此调用 va_arg 取精度值。若此时宽度值小于 0，则将
    // 字段精度值取为 0。
138                 /* get the precision */
139                 precision = -1;
140                 if (*fmt == '.') {
141                         ++fmt;
142                         if (is_digit(*fmt))
143                                 precision = skip_atoi(&fmt);
144                         else if (*fmt == '*') {
145                                 /* it's the next argument */   // 同上这里也应插入++fmt;
146                                 precision = va_arg(args, int);
147                         }
148                         if (precision < 0)
149                                 precision = 0;
150                 }
151
    // 下面这段代码分析长度修饰符，并将其存入 qualifer 变量。（h,l,L 的含义参见列表后的说明）。
152                 /* get the conversion qualifier */
153                 qualifier = -1;
154                 if (*fmt == 'h' || *fmt == 'l' || *fmt == 'L') {
155                         qualifier = *fmt;
156                         ++fmt;
157                 }
```

```
158
    // 下面分析转换指示符。
159                 switch (*fmt) {
    // 如果转换指示符是'c'，则表示对应参数应是字符。此时如果标志域表明不是左靠齐，则该字段前面
    // 放入'宽度域值-1'个空格字符，然后再放入参数字符。如果宽度域还大于 0，则表示为左靠齐，则在
    // 参数字符后面添加'宽度值-1'个空格字符。
160                 case 'c':
161                         if (!(flags & LEFT))
162                                 while (--field_width > 0)
163                                         *str++ = ' ';
164                         *str++ = (unsigned char) va_arg(args, int);
165                         while (--field_width > 0)
166                                 *str++ = ' ';
167                         break;
168
    // 如果转换指示符是's'，则表示对应参数是字符串。首先取参数字符串的长度，若其超过了精度域值，
    // 则扩展精度域=字符串长度。此时如果标志域表明不是左靠齐，则该字段前放入'宽度值-字符串长度'
    // 个空格字符。然后再放入参数字符串。如果宽度域还大于 0，则表示为左靠齐，则在参数字符串后面
    // 添加'宽度值-字符串长度'个空格字符。
169                 case 's':
170                         s = va_arg(args, char *);
171                         len = strlen(s);
172                         if (precision < 0)
173                                 precision = len;
174                         else if (len > precision)
175                                 len = precision;
176
177                         if (!(flags & LEFT))
178                                 while (len < field_width--)
179                                         *str++ = ' ';
180                         for (i = 0; i < len; ++i)
181                                 *str++ = *s++;
182                         while (len < field_width--)
183                                 *str++ = ' ';
184                         break;
185
    // 如果格式转换符是'o'，表示需将对应的参数转换成八进制数的字符串。调用 number()函数处理。
186                 case 'o':
187                         str = number(str, va_arg(args, unsigned long), 8,
188                                 field_width, precision, flags);
189                         break;
190
    // 如果格式转换符是'p'，表示对应参数是一个指针类型。此时若该参数没有设置宽度域，则默认宽度
    // 为 8，并且需要添零。然后调用 number()函数进行处理。
191                 case 'p':
192                         if (field_width == -1) {
193                                 field_width = 8;
194                                 flags |= ZEROPAD;
195                         }
196                         str = number(str,
197                                 (unsigned long) va_arg(args, void *), 16,
198                                 field_width, precision, flags);
199                         break;
```

```
200
    // 若格式转换指示是'x'或'X'，则表示对应参数需要打印成十六进制数输出。'x'表示用小写字母表示。
201                 case 'x':
202                         flags |= SMALL;
203                 case 'X':
204                         str = number(str, va_arg(args, unsigned long), 16,
205                                 field_width, precision, flags);
206                         break;
207
    // 如果格式转换字符是'd','i'或'u'，则表示对应参数是整数，'d', 'i'代表符号整数，因此需要加上
    // 带符号标志。'u'代表无符号整数。
208                 case 'd':
209                 case 'i':
210                         flags |= SIGN;
211                 case 'u':
212                         str = number(str, va_arg(args, unsigned long), 10,
213                                 field_width, precision, flags);
214                         break;
215
    // 若格式转换指示符是'n'，则表示要把到目前为止转换输出字符数保存到对应参数指针指定的位置中。
    // 首先利用 va_arg()取得该参数指针，然后将已经转换好的字符数存入该指针所指的位置。
216                 case 'n':
217                         ip = va_arg(args, int *);
218                         *ip = (str - buf);
219                         break;
220
    // 若格式转换符不是'%'，则表示格式字符串有错，直接将一个'%'写入输出串中。
    // 如果格式转换符的位置处还有字符，则也直接将该字符写入输出串中，并返回到 107 行继续处理
    // 格式字符串。否则表示已经处理到格式字符串的结尾处，则退出循环。
221                 default:
222                         if (*fmt != '%')
223                                 *str++ = '%';
224                         if (*fmt)
225                                 *str++ = *fmt;
226                         else
227                                 --fmt;
228                         break;
229                 }
230         }
231         *str = '\0';            // 最后在转换好的字符串结尾处添上 null。
232         return str-buf;         // 返回转换好的字符串长度值。
233 }
234
```