Unix Curses 库导论

Norman Matloff

http://heather.cs.ucdavis.edu/~matloff/

原版地址：http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.pdf

加州大学戴维斯分校计算机科学系

翻译：Mark

goonyangxiaofang@163.com

目录

1 历史

1.1 Curses 库的目的

许多被广泛使用的程序需要有一个终端光标移动功能。比如 vi (或者 vim 变种) 编辑器，它的许多功能都需要这样的功能。例如，当输入 j 键的时候，光标会移动到上一行；输入 dd 当前行会被删除，下面的行都向上移动一行，上面的行保持不变。

不同的终端有不同类型的光标动作，这样导致了一个潜在的问题。例如，如果一个程序想在 VT100 终端上向上移动一行，这个程序需要发送 Escape、[和 A 字符。

printf("%c%c%c", 27, '[', 'A');

(Escape 键的 ASCII 值码为 27)。但是对于 Televideo 920C 终端，程序必须发送 ctrl-K 字符，它的 ASCII 值为 11 。

printf("%c", 11);

很明显，像 vi 这样的程序的作者为了适应各种终端将变的疯狂和更糟。其他的每一个需要光标运动的程序员需要重新发明轮子，和 vi 作者做的工作一样，这样将会浪费很多时间。

这就是开发 curses 库的原因。它的目的就是为了减轻那些面向不同的终端需要写不同的代码的程序员的减轻。程序只需要去调用库，库可以知道针对什么样的终端具体做什么。

例如，如果你需要清屏，这里就不需要直接使用上面说到的任何字符序列。相反地，你只需要调用

clear();

curses 可以帮助这个程序做具体的工作，例如可以输出字符 Escape、[和 A ，以清屏。

1.2 Curses 角色的演化

在 Unix 软件的演化进程中，curses 库的发展是重要的一步。即便 VT100 终端成为了标准，curses 库继续扮演着重要的角色，它为程序员提供了一个抽象层，使其避免了解具体的 VT100 的光标动作代码。

Curses 库在今天的面向 GUI 世界里一直起着重要的作用，因为在很多应用中，使用键盘要比使用鼠标更为方便（许多 Microsoft Windows 应用程序里都提供了键盘快捷键）。今天，物理终端被很少使用了，但是典型的 Unix 工作包含了一些效仿 VT100 终端的 文本窗口。Vi 编辑器和其他基于 curses 的应用程序继续很流行。

2 包含和库文件

为了使用 curses ，你必须在你的源代码中包含这条语句

```
#include <curses.h>
```

你必须链接 curses 库

```
gcc –g sourcefile.c –lcurses
```

3 两个例子

在例子中学习

试着去运行这些程序！你不键入这些源代码，相反地，你可以获取从产生这个文档的生文件中获取它们，http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.tex，然后将非代码删除。

如果你比较急，你可以直接去第二个例子，有更好的注释并且使用了更多的 curses 特性。

3.1 一个简单的，快速引导的例子

第一个例子几乎没做什么。你可以认为这是一个原始的乏味的游戏。但是它给我们一个开头。

源文件最上面的注释告诉了你这个游戏做什么。编译并运行这个程序，输入你喜欢的字符。然后参考解释的注释阅读代码（从 main() 函数开始，一般情况下你应该如此）。

```
1   // 简单的 curses 例子；画输入的字符，按照列向下的方式，当到达最后一行时转向
2   // 右，当最右边到达时环绕。
3
4   #include <curses.h>
5
6   int r, c; // 当前行与列
7   int nrows, ncols; // 窗口的行列数
8
9   void draw(char dc)
10  {
11      move(r, c); // 移动到当前行和列
12      delch();
13      insch(dc); // 替换字符
14      refresh(); // 更新屏幕
15      ++r; // 下一行
16      // 检查是否需要转向右或是环绕
17      if (r == nrows)
18      {
19          r = 0;
20          ++c;
21          if (c == ncols)
22          {
23              c = 0;
24          }
```

```
25          }
26  }
27
28  int main()
29  {
30          int i;
31          char d;
32          WINDOW* wnd;
33
34          wnd = initscr(); //  初始化窗口
35          cbreak();  //  为键入键设置不要等
36          noecho(); //  设置不回送
37          getmaxyx(wnd, nrows, ncols);        //窗口的大小
38          clear();    //  清屏，设置光标到  (0, 0)
39          refresh(); //  实现从上次刷新所有改变
40
41          r = 0;
42          c = 0;
43          while (1)
44          {
45                  d = getch();     //  键盘输入
46                  if (d == 'q')
47                  {
48                          break;
49                  }
50                  draw(d);
51          }
52          endwin(); //  还原原始窗口和离开
53  }
```

3.2 第二个，更有用的例子

这个程序你可以实际使用。如果你需要结束掉一些进程，这个程序可以允许浏览和删除你想删的进程。

同样，编译和运行这个程序，结束一些你已经用于其他目的已创建的垃圾进程。（特别地，你可以删除 psax 它自身。）然后依据解释的注释阅读该代码。

```
1   // psax.c
2
3   // 运行 shell 命令 'ps ax', 在最后行显示它的输出，尽可能多的窗口适合；允许用户在窗口中上下移动，可以选择结束那些高亮的进程。
```

```
4
5   // 用法： psax
6
7   // 用户命令
8
9   // 'u': 上移
10  // 'd': 下移
11  // 'k': 删除高亮的进程
12  // 'r': 重新运行 'ps ax' 更新
13  // 'q': 结束
14
15  // 可能的扩展：允许滚动，这样可以使用户查看所有的 'ps ax' 输出，而不只是最后行；
16  // 允许卷上长行；
17  // 询问用户是否确认结束一个进程
18
19  #define MAXROW 1000
20  #define MAXCOL 500
21
22  #include <curses.h>
23
24  WINDOW* scrn; // 指向 curses 窗口对象
25
26  char cmdoutlines[MAXROW][MAXCOL]; // 'ps ax' 的输出，最好使用 malloc()
27
28  int ncmdlines; // cmdoutlines 的行数
29  int nwinlines;  // xterm 或 equiv. 窗口中 'ps ax' 输出的行数
30  int winrow;          // 屏幕上当前行的位置
31  int cmdstartrow;    // 显示的 cmdoutlines 的第一行的索引
32  int cmdlastrow;     // 显示的 cmdoutlines 的最后一行的索引
33
34  // 在 winrow 上用黑体重写
35  void highlight()
36  {
37      int clinenum;
38      attron(A_BOLD);
39
40      clinenum = cmdstartrow + winrow;
41
42      mvaddstr(winrow, 0, cmdoutlines[clinenum]);
43      attroff(A_BOLD);
44      refresh();
45  }
46
47  void runpsax()
```

```
48  {
49      FILE* p;
50      char ln[MAXCOL];
51      int row, tmp;
52      p = popen("ps ax", "r");
53      for (row = 0; row < MAXROW; ++row)
54      {
55          tmp = fgets(ln, MAXCOL, p);
56          if (tmp == NULL)
57          {
58              break;
59          }
60          strncpy(cmdoutlines[row], ln, COLS);
61          cmdoutlines[row][MAXCOL - 1] = 0;
62      }
63      ncmdlines = row;
64      close(p);
65  }
66
67  void showlastpart()
68  {
69      int row;
70      clear();
71
72      if (ncmdlines <= LINES)
73      {
74          cmdstartrow = 0;
75          nwinlines = ncmdlines;
76      }
77      else
78      {
79          cmdstartrow = ncmdlines - LINES;
80          nwinlines = LINES;
81      }
82      cmdlastrow = cmdstartrow + nwinlines - 1;
83
84      for (row = cmdstartrow, winrow = 0; row <= cmdlastrow; ++row, ++winrow)
85      {
86          mvaddstr(winrow, 0, cmdoutlines[row]);
87      }
88      refresh();
89      --winrow;
90      highlight();
91  }
```

```
92
93  void updown(int  inc)
94  {
95          int tmp = winrow + inc;
96          if (tmp >= 0 && tmp < LINES)
97          {
98                  mvaddstr(winrow, 0, cmdoutlines[cmdstartrow  + winrow]);
99                  winrow = tmp;
100                 highlight();
101         }
102 }
103
104 void rerun()
105 {
106         runpsax();
107         showlastpart();
108 }
109
110 void prockill()
111 {
112         char* pid;
113         pid = strtok(cmdoutlines[cmdstartrow  + winrow], " ");
114         kill(atoi(pid),  9);
115         rerun();
116 }
117
118 int main()
119 {
120         char c;
121         scrn = initscr();
122         noecho();
123         cbreak();
124         runpsax();
125         showlastpart();
126         while (1)
127         {
128                 c = getch();
129                 if (c == 'u')
130                 {
131                         updown(-1);
132                 }
133                 else if (c == 'd')
134                 {
135                         updown(1);
```

```
136             }
137             else if (c == 'r')
138             {
139                     rerun();
140             }
141             else if (c == 'k')
142             {
143                     prockill();
144             }
145             else
146             {
147                     break;
148             }
149     }
150     endwin();
151     return 0;
152 }
```

3.3 例子中 Coded 和 Raw 模式注释

在你写的大多数程序中，键盘操作是出于熟模式下的。也就是说，在你按下回车键之前你的键盘输入是不会发送给程序的（例如，不会发送给 scanf() 函数或 cin ）。这种方式可以允许你使用退格键来删除那些你键入的但是又想撤销字符。并且，你的程序不能识别到你删除的字符以及退格键（其 ASCII 码为 8）。

记住，当你在键盘上敲击字符时，字符被送到操作系统。操作系统一般情况下将这些字符传递给你的应用程序（例如，传递给调用接口 scanf() 或者 cin ），但是如果操作系统发现了退格键字符，操作系统不会将这个字符传递给应用程序。事实上，操作系统在用户键入回车键之前不会将任何字符传递给程序。

另一种是原生模式。这种模式下，操作系统将每个字符传递给应用程序。如果用户键入了退格键，它被看作与其他字符一样，没有什么特殊的操作。应用程序接收到一个 ASCII 码为 8 的字符，这由程序员决定如何处理这个字符。

你可以调用 cbreak() 函数从熟模式切换到原生模式，也可以调用 nocbreak() 函数从原生模式切换到熟模式。

相似地，默认模式是为了操作系统来回显字符。如果用户键入 A 键（没有 Shift），操作系统将在屏幕上打印出‘a’。有些情况使我们不需要回显字符，例如在我们意见看到的例子中，或者在输入密码的情况下。我们可以通过调用 noecho() 函数关闭回显功能，也可以调用 echo() 函数恢复回显功能。

4 重要的调试笔记

不用使用 printf() 或者 cout 来调试！确保你使用调试工具，例如 GDB 或者带有 DDD 界面的 GDB 。如果在日常编程工作中不适用一个调试工具，你将花费大量的不必要的时间和碰到很多的挫折。请看我的调试幻灯片演示：
http://heather.cs.ucdavis.edu/~matloff/debug.html

在 curses 程序中，你无法使用 printf() 或 cout 来调试程序，因为那种将会把你的程序输出弄的一团糟。所以一个调试工具是必要的。这里我们使用 GDB 和 DDD 来调试 curses ，这些调试工具在 Unix 世界里非常常用。你使用这个程序将你的调试信息输出与你的 curses 应用程序输出分开。这里我将演示如何调试。

4.1 GDB

在文本窗口中启动 GDB 。为你的 curses 应用程序选择另一个窗口来运行，为后面的窗口选择一个设备名字（我们可以叫做“执行窗口”）。在窗口中运行 tty Unix 命令。在这个例子中，我们假设命令输出是 “/dev/pts/10” 。在 GDB 中的命令是：
(gdb) tty /dev/pts/10

在 run 命令之前，我们还要做其他事情。在执行窗口中键入：
Sleep 10000

Unix 的 sleep 命令让 shell 在给定的时间内进入失效状态，在这个例子中是 10 秒。这个是很必要的，因为这样可以使我们在窗口中键入的信息多传递给我们的应用程序而不是传递给了 shell 。

现在回到 GDB 并且执行 run 命令。记住，不管何时你的程序执行到了断点并且从键盘输入，你必须在执行窗口中键入（你也将要看到程序的输出）。把这些做完后，在执行窗口中键入 ctrl-C 删除 sleep 命令，确保 shell 可用。

注意如果有什么错误并且你的程序提前结束了，执行窗口可以保持一些非标准的终端配置，不如 cbreak 模式。修改这个，回到窗口并键入 ctrl-j ‘reset’ 再一次键入 ctrl-j 。

4.2 DDD

在 DDD 差不多。只是点击视图|执行窗口，一个弹出来的窗口作为执行窗口。

5 一些主要的 Curses APIs，属性和环境变量

5.1 环境

- 窗口中行和列的的号码从 0 开始，从上到下从左到右。所以左上角的最表是 (0, 0)。
- LINES, COLS
  窗口中行和列的数目。

5.2 APIs

（许多 API 是宏而不是真正的函数）
这里有一些你会调用的 curses 函数。注意这里罗列的函数中只有一些是可以使用的。可以使用 curses 做许多其他的东西，例如子窗口，窗体样式输入等。第六部分有对资源的介绍。

- **WINDOW* initsrc()**
  REQUIRED. 为 curses 初始化整个屏幕。返回一个执行 WINDOW 类型结构体的指针，该指针被其他函数所使用。
- **endwin()**
  重新设置终端，例如恢复回显功能、熟模式等。
- **cbreak()**
  设置终端，键入的字符即是所读取的字符，不用等待键入回车键。退格键和其他控制键（包括回车键）失去了他们原来的意义。
- **nocbreak()**
  回到一般模式
- **noecho()**
  关闭将输入字符显示到屏幕上的回显功能。
- **echo()**
  恢复回显功能
- **clear()**
  清屏，从新将光标设置在左上角。
- **move(int, int)**
  将光标设置到指定的列和行。
- **addch(char)**
  在当前光标位置打印给定的字符，改写这个位置之前的字符，将光标移到右面的下一个位置。
- **insch(char)**
  与 addch() 一样，插入代替改写，所有右边的字符都向右移动一个位置。
- **mvaddstr(int, int, char*)**
  将光标移动到指定的行和列，在指定位置打印字符串。
- **refresh()**
  更新屏幕，以相应上一次调用这个函数所发生的所有变化。不过我们做了什么变化，例如调用上面的 addch() 或者不会出现在屏幕上的变化，只有调用了 refresh() 才回有效

果。

- **delch()**

  将当前光标处的字符删除掉，右边的所有字符都将向做移动一个位置，光标的位置不会变化。

- **int getch()**

  从键盘处读取一个字符。

- **char inch()**

  返回当前光标下的字符。

- **getyx(WINDOWS*, int, int)**

  返回窗口的光标当前位置的行和列数。

  （注意，这里函数是宏，所以这里是 ints 不是只想 int 的指针）

- **getmaxyx(WINDOW*, int, int)**

  返回指定窗口的行和列数。

- **scanw(), printw()**

  curses 环境下的类似与 scanf() 和 printf() 的函数。避免在 curses 环境下使用 scanf() 和 printf() 函数，否则将会造成奇异的结果。注意 printw() 和 scanw() 函数重复调用 addch(), 所以他们是做改写工作不是插入。Scanw 知道遇到换行符或者回车符的时候才结束，wgetstr() 函数也是如此。

- **attron(const), attroff() const**

  将给定的属性开启或者关闭。

5.3 属性

字符可以通过许多不同的方式来显示，比如一般模式（A_NORMAL）和 黑体模式（A_BOLD）。可以通过 APIs attron() 和 attroff() 函数设置。前面的被调用后，所有打印在屏幕上的内容使用给定参数的属性，后面的被调用则终止这种状态。

6 进一步学习

在 man 手册中有基本的帮助教程。

man curses

(你可以用 ncurese 代替 curses) 个别的函数有单独的 man 页面。

在网上有很多好的辅导资料。在你最喜爱的搜索引擎中键入“curses tutorials”或者“ncurses tutorials”。以可以从这个辅导材料开始：
http://www.linux.com/howtos/NCURSES-Programming-HOWTO/index.html

注意，尽管有一些关于非 C 语言的辅导资料，例如 Perl 或者 Python 的。我不推荐这些（包括我自己，Python）。不仅语法存在不同，还有 API 也有很多的不同。