第10章 事件驱动编程

实验目的

- 了解curses函数和curses函数库
- 学习屏幕管理、使用定时器和信号实现进程的并发执行
- 学会异步事件驱动编程
- 利用上述知识编写一个视频动画游戏

主要内容

- 背景知识
  - curses库概述
  - curses库编程
- 实验内容
  - 利用curses库实现弹球游戏
  - 利用多线程实现弹球游戏

curses函数库

- 控制字符输入/输出的格式
- termios缺点，转义处理
- curses优点
  - 提供与终端无关的字符处理方式
  - 可以管理键盘
  - 支持多窗体管理

curses库的使用

- curses *vs.* ncurses
- 源文件包含头文件curses.h
- 编译时加 –lcurses选项
  - gcc program.c –o program –lcurses
  - gcc –I/usr/include/ncurses program.c –o program –lncurses
- curses配置情况的检查
  - 查看头文件：ls –l /usr/include/*curses.h
  - 查看库文件：ls –l /usr/lib/*curses*

curses相关基本概念

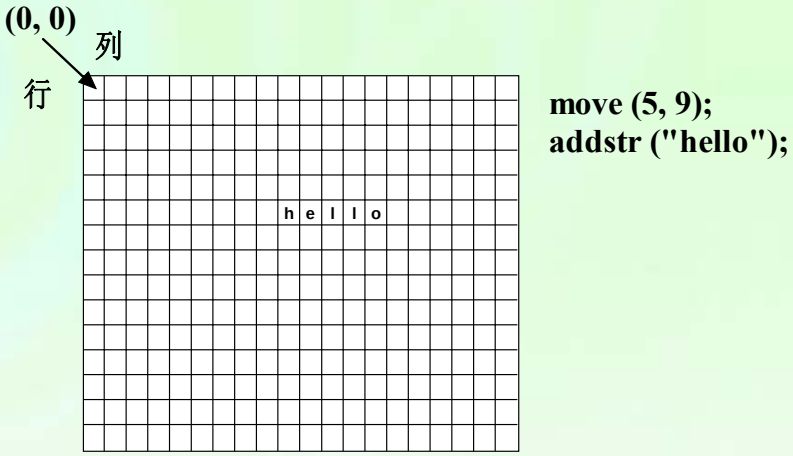
- cureses工作于屏幕、窗口和子窗口上
  - 屏幕：正在写的设备，占据设备上全部的可用显示面积
  - 窗口
    - ✓curses窗口(称为stdscr)：至少存在一个，与物理屏幕的尺寸相同
    - ✓其他窗口：尺寸小于屏幕窗口，可以重叠
    - ✓子窗口：必须包含在父窗口内

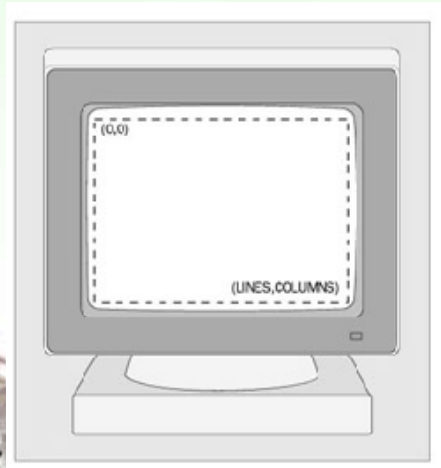
curses库的核心数据结构

- **stdscr**
  - 对应于“标准屏幕”，是curses程序的默认输出窗口
  - 工作原理与stdio函数库中的stdout非常相似
  - 在curses函数产生输出时被刷新
  - 在调用refresh函数之前，输出到stdscr上的内容不会在屏幕上显示
- **curscr**
  - 对应当前屏幕的样子
  - 调用refresh函数时，curses函数库会比较stdscr及curscr之间的不同之处，然后用这个两个数据结构之间的差异来刷新屏幕
  - Curses程序需要了解stdscr，但不需要使用curscr数据结构

curses程序中输出字符的过程

- 用curses函数刷新逻辑屏幕
- 用refresh函数刷新物理屏幕
- 逻辑屏幕
  - 通过字符数组来实现
  - 屏幕左上角—坐标(0, 0)为起点
  - 坐标形式
    - ✓y在前，表示行号
    - ✓x在后，表示列号
  - 每个位置包括该屏幕位置的字符及其属性





curses中的全局变量

- WINDDW* curscr：当前屏幕
- WINDOW* stdscr：标准屏幕
- int LINES：终端上的行数
- int COLS：终端上的列数
- bool TRUE：真标志，1
- bool FALSE：假标志，0
- int ERR：错误标志，－1
- int OK：OK标志，0

curses程序结构

- 使用curses函数库之前需做初始化
  - initscr函数
- 使用过程中会创建和删除一些临时数据结构
- 结束后需恢复到原先设置
  - endwin函数

主要内容

- 背景知识
  - curses库概述
  - curses库编程
- 实验内容
  - 利用curses库实现弹球游戏
  - 利用多线程实现弹球游戏

简单curses程序

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

int main() {
    initscr();

    move(5, 15);
    printw("%s", "Hello world");
    refresh();

    sleep(2);

    endwin();
    exit(EXIT_SUCCESS);
}
```

初始化和重置函数

- 函数定义
  - #include <curses.h>
  - WINDOW *initscr(void);
    - 在一个程序中只能调用一次
    - 判断终端类型和初始化Curses数据结构，同时也对终端进行一次刷新以清除屏幕，为以后的操作做准备
    - 成功时，返回一个指向stdscr结构的指针
    - 失败时，返回一个诊断信息并使程序结束
  - int endwin(void);
    - 将恢复tty终端原来的状态，把光标移到屏幕的左下角,重置终端为正确的非虚拟模式
    - 调用成功时返回OK，失败时返回ERR
  - 清屏处理方法
    - 调用endwin函数退出curses
    - 调用clearok(stdscr, 1)
    - 调用refresh函数

refresh函数

- 函数原形
  - int refresh(void);
- 说明
  - curses最常用的一个函数
  - 在调用屏幕输出函数试图改变萤幕上的画面时，curses并不会立刻对屏幕做改变，而是等到refresh()调用后，才将刚才所做的变动一次完成。
  - 其余信息维持不变，以尽可能送最少字符发送至屏幕上，减少屏幕重绘时间。
  - 如果是initscr()后第一次调用refresh()，curses将做清除屏幕的工作

基本屏幕字符输出处理函数

- **int addch(const chtype ch);**
  - 在当前光标位置输入单个字符，并将光标右移一位。可以附加加字符修饰参数的一类函数
- **int addchstr(chtype *const string_to_add);**
  - 在当前光标位置输入一个字符串，可以附加加字符修饰参数的一类函数
- **int printw(char *format, ...);**
  - 采用与printf相同机制字符串格式化，并将其添加到光标的当前位置
- **int insch(chtype ch);**
  - 把字符ch插入到光标的左边，光标后面的所有字符则向右移动一个位置。
  - 在这一行最右端的字符可能会丢失
- **int insertln(void);**
  - 插入一个空行，将现有行依次向下移一行

基本屏幕字符输出处理函数

- **int delch(void);**
  - 删除光标左边的字符，并把光标右边余下的字符向左移动一个位置
- **int deleteln(void);**
  - 删除光标下面的一行，并把下面所有的其他行都向上移动一个位置。 此外，屏幕最底下的一行将被清除
- **int box(WINDOW *win_ptr, chtype vertical_char, chtype horizontal_char);**
  - 自动画方框
  - vertical_char : 画方框时垂直方向所用字符
  - horizontal_char: 画方框时水平方向所用字符
- **int beep(void);**
  - 让程序发出声音
- **int flash(void);**
  - 使屏幕闪烁

从屏幕读取基本函数

- **chtype inch(void);**
  - 返回光标当前位置的字符及其属性
- **int instr(char *string);**
  - 将返回内容写到字符数组中
- **int innstr(char *string, int number_of_characters);**
  - 将返回内容写到字符数组中，可以指定返回字符的个数

清除屏幕

- **int erase(void);**
  - 在每个屏幕位置写上空白字符
- **int clear(void);**
  - 与erase()类似，也是清屏，但通过调用clearok函数来强制重现屏幕原文
  - clearok函数会强制执行清屏操作，并在下次调用refresh函数时重现屏幕原文
- **int clrtobot(void);**
  - 清除当前光标所在行下面的所有行，包括当前光标所在行中的光标位置右侧直到行尾的内容
- **int clrtoeol(void);**
  - 清除当前光标所在行中光标位置右边至行尾的内容

移动光标

- **int move(int y, int x);**
  - 将光标移动至(y, x)的位置
  - LINES和COLUMNS决定了y与x的最大取值
  - 本函数不会使物理光标移动，仅改变逻辑屏幕上的光标位置，下次的输出内容将出现在该位置上
  - 如果希望物理屏幕上的光标位置在调用move后立即变化，需要立即调用refresh函数
- **int leaveok(WINDOW *window_ptr, bool leave_flag);**
  - 设置标志，用于控制在屏幕刷新后curses将物理光标放置的位置
  - 默认情况下，该标志为false，意味屏幕刷新后，硬件光标将停留在屏幕上逻辑光标所处的位置
  - 如果该标志设为true，则硬件光标会被随机地放在屏幕上的任意位置

字符属性

- 控制字符在屏幕上的显示方式
- 前提是用于显示的硬件设备能够支持要求的属性
- 属性定义
  - A_UNDERLINE：加底线
  - A_REVERSE：反白
  - A_BLINK：闪烁
  - A_BOLD：高亮度
  - A_NORMAL：标准模式(只能配合attrset()使用)
  - A_DIM：半亮显示
  - A_STANDOUT：终端字符最亮

字符属性控制函数

- **int attron(chtype attribute);**
  - 开启某一种特殊属性模式
  - 只从被调用的地方开始设置
- **int attroff(chtype attribute);**
  - 关闭某一种特殊属性模式
- **int attrset(chtype attribute);**
  - 把当前窗口设置为参数attrs所指定的属性
  - 为整个窗口设置一种修饰属性，会覆盖掉先前为整个窗口设置的任何修饰属性
- **int standout(void);**
  - 更加通用的强调或者“突出”显示模式
  - 在大多数终端上，通常映射成反白显示
- **int standend(void);**
  - 关闭所有设置的修饰
  - 这个函数的作用和attrset(A_NORMAL)函数是相同的

字符属性操作示例1

- 读取文件中每个字符并寻找有"/*“
- 一旦找到，就会调用attron()函数开始为输出文字加粗加亮
- 当找到"*/"（注释结束处标志）的地方，就会使用attroff()函数关闭修饰效果

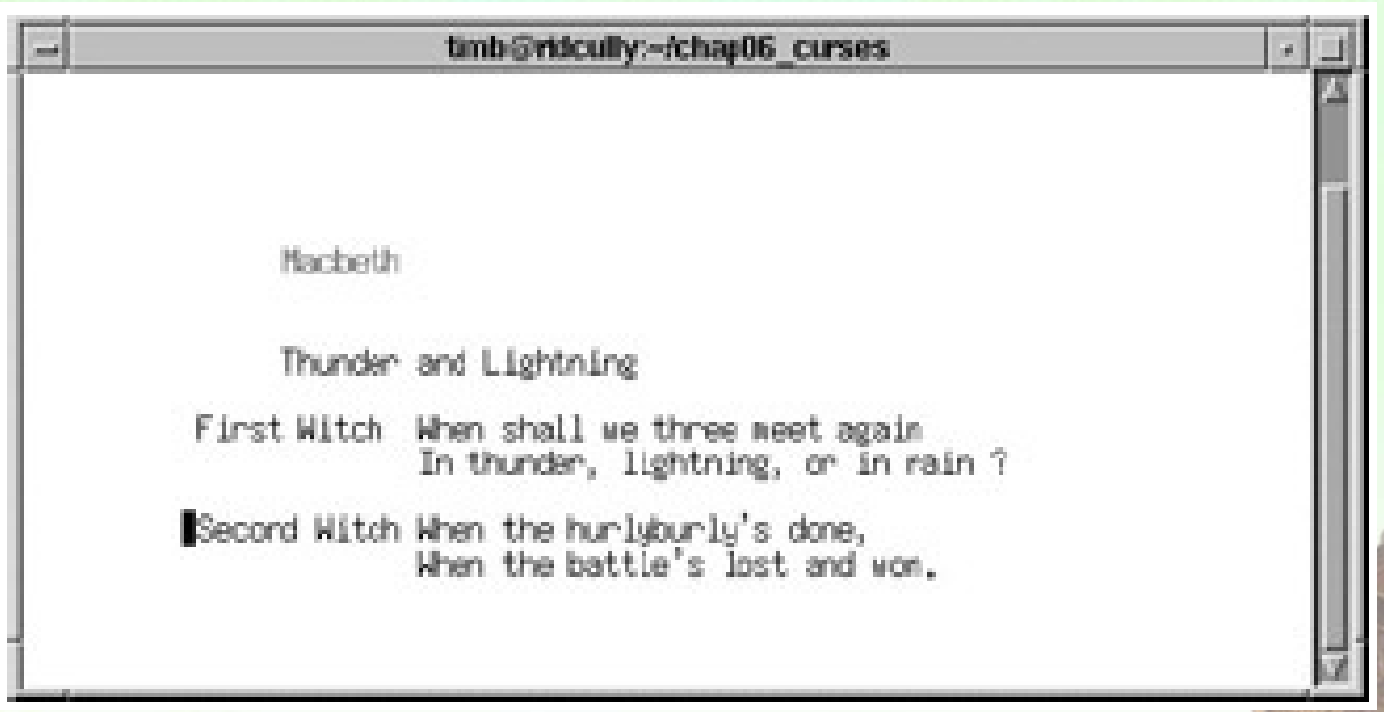
字符属性操作示例1

```
#include <ncurses.h>

int main(int argc, char *argv[]){
   int ch, prev;
   FILE *fp;
   int goto_prev = FALSE, y, x;
   if(argc != 2){
      printf("Usage: %s <a c file name>\n", argv[0]);
      exit(1);
   }
   fp = fopen(argv[1], "r");  /* 检测文件是否成功打开 */
   if(fp == NULL){
      perror("Cannot open input file");
      exit(1);
   }
   initscr(); /* 初始化并进入CURSES模式 */
   prev = EOF;
```

字符属性操作示例1

```
while((ch = fgetc(fp)) != EOF){
  /* 当读到字符"/"和"*"的时候调用开启修饰函数 */
  if(prev == '/' && ch == '*') {
    attron(A_BOLD); /* 将"/"和"*"及以后输出的文字字体加粗  */
    goto_prev = TRUE;
  }
  if(goto_prev == TRUE){ /* 回到"/"和"*"之前开始输出 */
    getyx(stdscr, y, x);
    move(y, x - 1);
    printw("%c%c", '/', ch);  /* 实际打印内容的部分 */
    ch = 'a';  /* 避免下次读取变量错误，这里赋一个任意值*/
    goto_prev = FALSE;  /* 让这段程序只运行一次 */
  } else
    printw("%c", ch);
  refresh(); /* 将缓冲区的内容刷新到屏幕上 */
  if(prev == '*' && ch == '/')
    attroff(A_BOLD);/* 当读到字符"*"和"/"的时候调用修饰关闭函数*/
  prev = ch;
}
getch();
endwin(); /* 结束并退出Curses模式 */
return 0;
}
```

移动、插入和属性操作示例



移动、插入和属性操作示例

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

int main()
{
    const char witch_one[] = " First Witch  ";
    const char witch_two[] = " Second Witch ";
    const char *scan_ptr;

    initscr();
```

移动、插入和属性操作示例

```
move(5, 15);
attron(A_BOLD);
printw("%s", "Macbeth");
attroff(A_BOLD);
refresh();
sleep(1);

move(8, 15);
attron(A_DIM);
printw("%s", "Thunder and Lightning");
attroff(A_DIM);
refresh();
sleep(1);
```

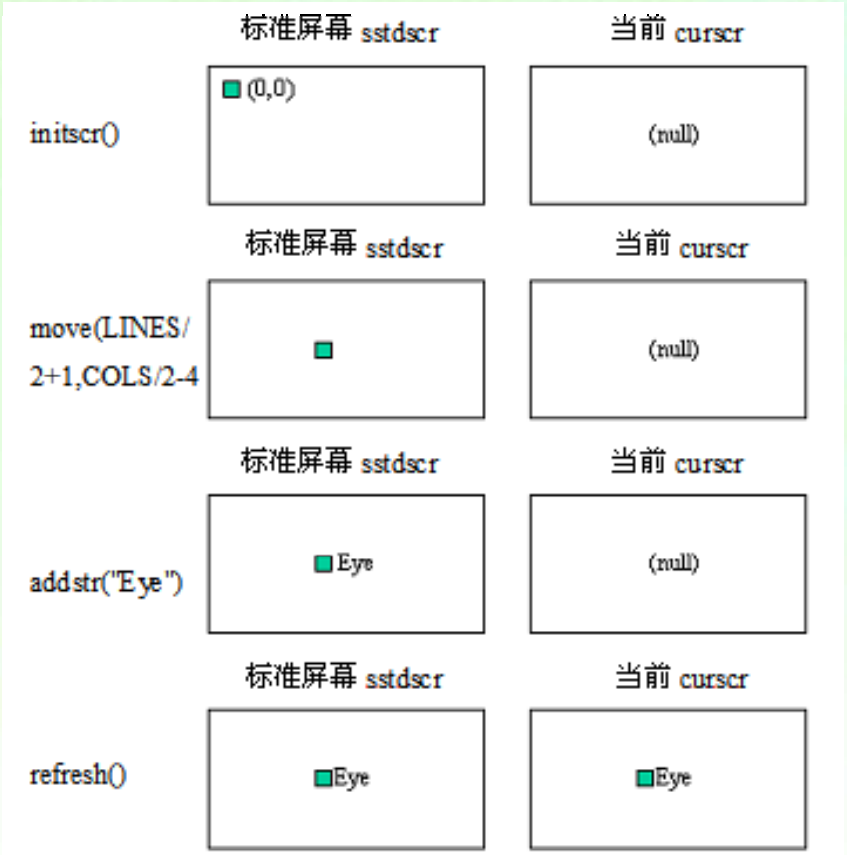
移动、插入和属性操作示例

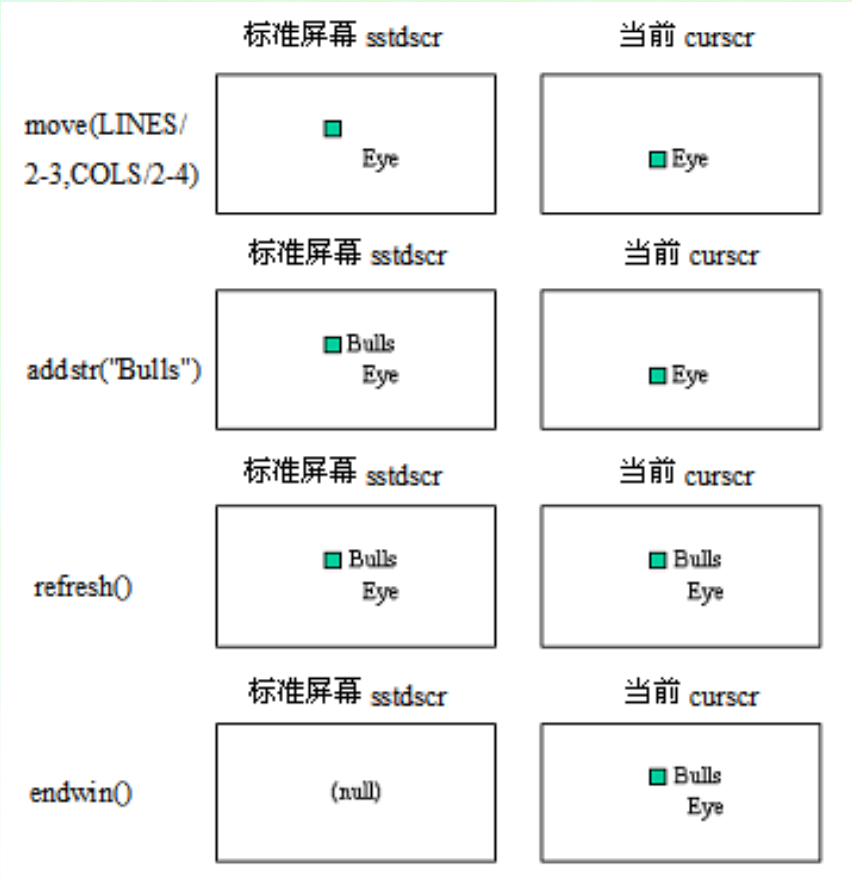
```
move(10, 10);
printw("%s", "When shall we three meet again");
move(11, 23);
printw("%s", "In thunder, lightning, or in rain ?");
move(13, 10);
printw("%s", "When the hurlyburly's done,");
move(14,23);
printw("%s", "When the battle's lost and won.");
refresh();
sleep(1);
```

移动、插入和属性操作示例

```
    attron(A_DIM);
    scan_ptr = witch_one + strlen(witch_one);
    while(scan_ptr != witch_one) {
        move(10,10);
        insch(*scan_ptr--);
    }

    scan_ptr = witch_two + strlen(witch_two);
    while (scan_ptr != witch_two) {
        move(13, 10);
        insch(*scan_ptr--);
    }
    attroff(A_DIM);

    refresh();
    sleep(1);

    endwin();
    exit(EXIT_SUCCESS);
}
```

各种操作对stdsrc及curscr的影响





curses库基本操作示例

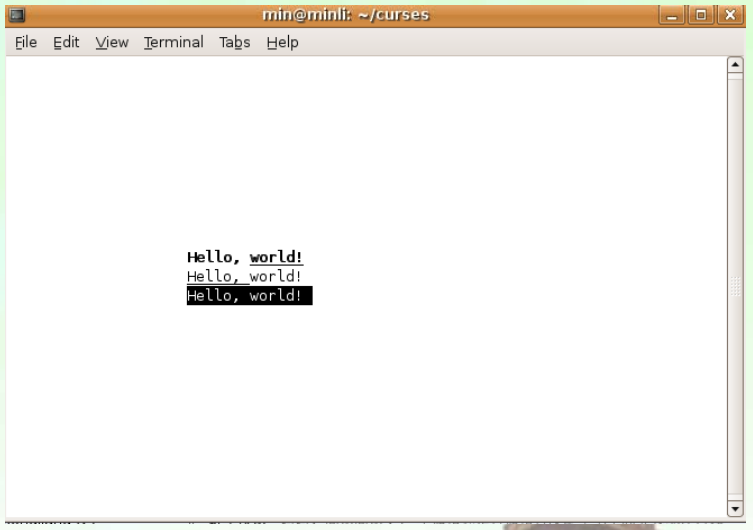
❖ cursestest.c

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

int main(int argc, char* argv[]) {
  initscr( );
  /*粗体显示*/
  move(10, 20);
  attron(A_BOLD);
  addstr("Hello, ");
  refresh( );
  sleep(1);
  /*粗体+下划线*/
  attron(A_UNDERLINE);
  addstr("world!");
  refresh( );
  sleep(1);
  /*下划线*/
  move(11,20);
  attroff(A_BOLD);
  addstr("Hello, ");
  refresh( );
  sleep(1);
```

curses库基本操作示例

❖ **cursestest.c**

```
    /*普通模式*/
    attroff(A_UNDERLINE);
    addstr("world!");
    refresh( );
    sleep(1);
    /*反白*/
    move(12,20);
    standout( );
    addstr("Hello, world!");
    standend( );
    refresh( );
    sleep(10);
    endwin( );
    return 0;
}
```



键盘工作模式

- int echo(void);
- int noecho(void);
  - 控制从键盘输入字元时是否将字元显示在终端机上
  - 系统预设是开启的
- int cbreak(void);
- int nocbreak(void);
  - 把终端的CBREAK模式打开或关闭
    - ✓ 如果CBREAK打开则程式就能即时使用读取的输入信息。
    - ✓ 如果CBREAK关闭，则输入将被缓存起来，直到产生新的一行
- int raw(void);
- int noraw(void);
  - 将把RAW模式打开或关闭
  - RAW模式不处理特别字符，此时不可以输入特殊字符序列来产生信号或进行流控制
  - noraw同时恢复行模式和特殊字符处理功能

键盘输入

- int getch(void);
  - 从键盘读取一个字符
  - 返回值是整数值
- int getstr(char *string);
  - 从键盘读取一串字符
- int getnstr(char *string, int number_of_characters);
  - 允许对读取的字符数目加以限制
- int scanw(char *format, ...);
  - 与scanf类似，从键盘读取一串字符

键盘输入示例

❖ **passwordtest.c**

```
#include <unistd.h>
#include <stdlib.h>
#include <curses.h>
#include <string.h>

int main(int argc, char *arg[]) {
  char name[20];
  char password[20];
  char *real_password = "123456";
  initscr( );
  move(5, 10);
  addstr("User name:");
  getnstr(name, sizeof(name)); /*获得输入*/
  move(7, 10);
  addstr("Password:");
  refresh( );
  cbreak( );          /*"中断模式"*/
  noecho( );          /*关闭回显*/
  memset(password, 0, sizeof(password));
  int len = sizeof(password);
```

键盘输入示例

❖ passwordtest.c

```
for (int i=0; i<len ; i++)  {
  password[i] = getch( );
  move(7, 20+i);
  addch('*');
  refresh( );
  if ( password[i] == '\n' )
    break;
  if ( strcmp(password, real_password)==0 )
    break;
}
echo( );/*打开回显*/
nocbreak( );
move(9, 10 );
if ( strcmp(password, real_password)==0 )
  addstr("Correct");
else
  addstr("Wrong");
refresh( );
endwin( );
return 0;
}
```

窗口

- 在屏幕上同时显示多个不同尺寸的窗口
- **WINDOW结构**
  - stdscr是其特例
  - WINDOW *newwin(int lines, int cols, int y, int x);
    - 通过屏幕位置(y, x)以及指定的行数ines和列数cols来创建一个新窗口
    - 成功时返回一个指向新窗口的指针，创建失败返回null
    - 如果希望将新窗口的右下角位于屏幕的右下角，则可以将行数或是列数指定为零
    - 所有的窗口必须适合当前屏幕，如果新窗口任何部分超出了屏幕区域，newwin函数就会失败。
  - int delwin(WINDOW *window_to_delete);
    - 删除一个由newwin所生成的窗口
    - 不要试图删除curses自己的窗口，stdscr与curscr

通用函数

❖ 函数原型

- int addch(const chtype char);
- int waddch(WINDOW *window, const chtype char)
- int mvaddch(int y, int x, const chtype char);
- int mvwaddch(WINDOW *window, int y, int x, const chtype char);
- int printw(char *format, ...);
- int wprintw(WINDOW *window, char *format, ...);
- int mvprintw(int y, int x, char *format, ...);
- int mvwprintw(WINDOW *window, int y, int x, char *format, ...);

❖ 说明

- 当添加w前缀时，一个额外的WINDOW指针必须添加到参数列表中。
- 当添加mv前缀时，两个额外的参数，y与x位置，必须添加到参数列表中。他们指明了执行操作的位置。
- y与x是相对于窗口的，而不是屏幕，(0,0)是窗口的左上角。

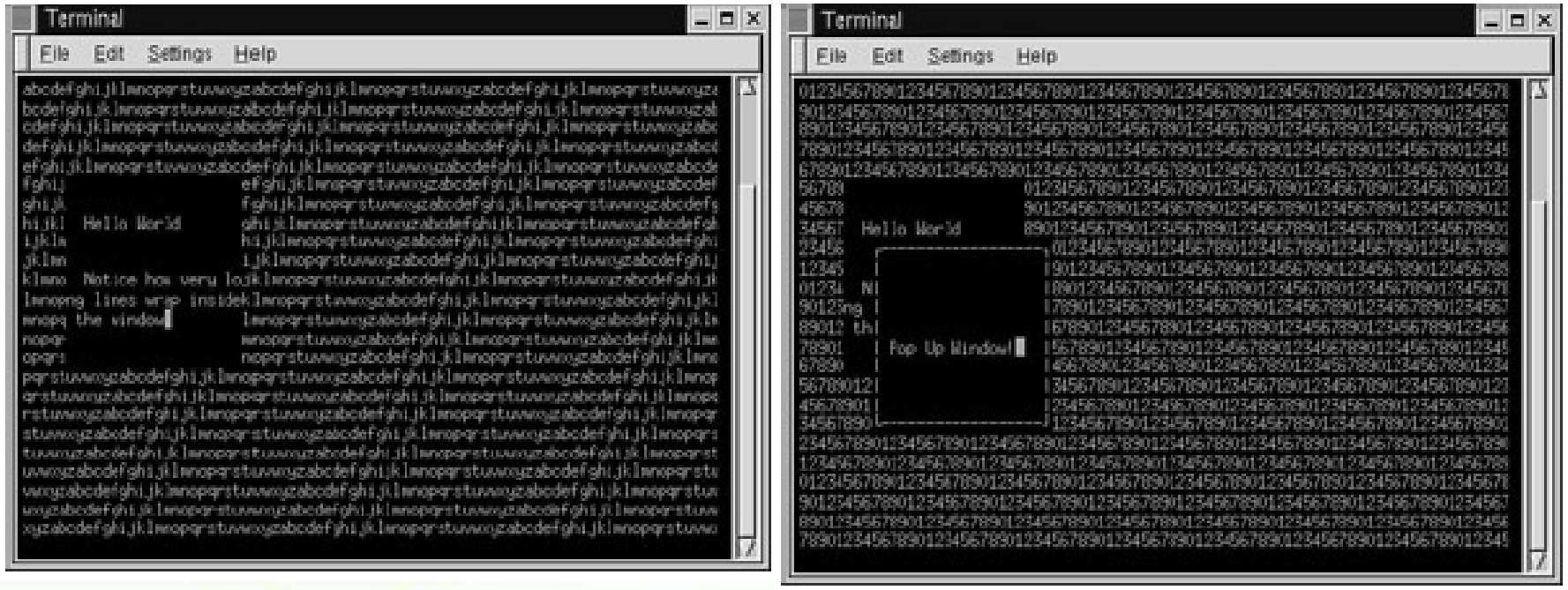
移动和更新窗口

- int mvwin(WINDOW *window, int y, int x);
  - 在屏幕上移动一个窗口
  - 如果将一个窗口的任何部分移出屏幕区域之外，mvwin函数就会失败。
- int wrefresh(WINDOW *window);
- int wclear(WINDOW *window);
- int werase(WINDOW *window);
  - 以上三个函数指向一个特定的窗口，而不是stdscr。
- int scrollok(WINDOW *window, bool scroll_flag);
  - 当传递一个布尔值true(通常为非零)时,会允许一个窗口的滚动。
  - 默认情况下，窗口并不可以滚动。
- int scroll(WINDOW *window);
  - 将窗口向上滚动一行

移动和更新窗口

- **int touchwin(WINDOW *window);**
  - 通知curses库其参数所指向的窗口内容已经发生了改变。
  - 这意味着curses库会在下一次调用wrefresh时重新绘制窗口，尽管我们并没有实际的改变窗口内容。
  - 当有多个窗口在屏幕上层叠显示而需要决定显示哪个窗口时，这个函数会非常有用。

多窗口示例


Terminal
File Edit Settings Help
Hello World
Notice how very lo
lines wrap inside
the windo
Terminal
File Edit Settings Help
Hello World
Pop Up Window!

多窗口示例

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

int main()
{
    WINDOW *new_window_ptr;
    WINDOW *popup_window_ptr;
    int x_loop;
    int y_loop;
    char a_letter = 'a';

    initscr();
```

多窗口示例

```
move(5, 5);
    printw("%s", "Testing multiple windows");
    refresh();

    for (y_loop = 0; y_loop < LINES - 1; y_loop++) {
        for (x_loop = 0; x_loop < COLS - 1; x_loop++) {
            mvwaddch(stdscr, y_loop, x_loop, a_letter);
            a_letter++;
            if (a_letter > 'z') a_letter = 'a';
        }
    }

    /* Update the screen */
    refresh();
    sleep(2);
```

多窗口示例

```
new_window_ptr = newwin(10, 20, 5, 5);
mvwprintw(new_window_ptr, 2, 2, "%s", "Hello World");
mvwprintw(new_window_ptr, 5, 2, "%s",
          "Notice how very long lines wrap inside the window");
wrefresh(new_window_ptr);
sleep(2);
```

多窗口示例

```
a_letter = '0';
for (y_loop = 0; y_loop < LINES -1; y_loop++) {
  for (x_loop = 0; x_loop < COLS - 1; x_loop++) {
      mvwaddch(stdscr, y_loop, x_loop, a_letter);
      a_letter++;
      if (a_letter > '9')
          a_letter = '0';
    }
}

refresh();
sleep(2);
```

多窗口示例

```
wrefresh(new_window_ptr);
sleep(2);
```

```
touchwin(new_window_ptr);
wrefresh(new_window_ptr);
sleep(2);
```

```
popup_window_ptr = newwin(10, 20, 8, 8);
box(popup_window_ptr, '|', '-');
mvwprintw(popup_window_ptr, 5, 2, "%s", "Pop Up Window!");
wrefresh(popup_window_ptr);
sleep(2);
```

多窗口示例

```
    touchwin(new_window_ptr);
    wrefresh(new_window_ptr);
    sleep(2);
    wclear(new_window_ptr);
    wrefresh(new_window_ptr);
    sleep(2);
    delwin(new_window_ptr);
    touchwin(popup_window_ptr);
    wrefresh(popup_window_ptr);
    sleep(2);
    delwin(popup_window_ptr);
    touchwin(stdscr);
    refresh();
    sleep(2);
    endwin();
    exit(EXIT_SUCCESS);
}
```

优化屏幕刷新

- 问题
  - 刷新多个窗体需要一些繁琐，在一个慢速的链接上，屏幕的绘制相当的慢
  - 目标就是要尽量减少要在屏幕上的绘制的字符数
- 基本函数
  - int wnoutrefresh(WINDOW *window_ptr);
    - ✓ 决定哪些字符需要发送到屏幕，但是并不实际的发送
  - int doupdate(void);
    - ✓ 向终端发送实际的改变
- 说明
  - 如果只是简单地调用wnoutrefresh，其后立即调用doupdate，其效果就如同调用wrefresh一样
  - 如果希望重新绘制一个窗体栈，可以在每一个窗体(当然需要以正确的顺序)上调用wnoutrefresh函数，然而在最后一个wnoutrefresh函数之后调用doupdate函数
  - 这使得curses按顺序在每一个窗体上执行屏幕更新计算，并且只输出更新的屏幕。这会使得curses尽量减少需要发送的字符数

子窗体

- **WINDOW *subwin(WINDOW *parent, int lines, int cols, int y, int x);**
  - subwin 函数具有与newwin几乎相同的参数列表
  - 子窗体的删除方式也与其他的窗体使用一个delwin调用方式相同
  - 与新窗体类似，可以使用一系列的 mvw函数将数据写入子窗体中。但是却有一点重要的区别
    - ✓子窗体本身并不会存储一个单独的屏幕字符集，他们与子窗体创建时所指定的父窗体共享存储空间
    - ✓这就意味着子窗体中的任何改动也同时会发生在底层的父窗体中，所以当一个子窗体被删除时，屏幕并不会发生变化
  - int delwin(WINDOW *window_to_delete);

子窗体示例

- 在将sub_window_ptr指向subwin的调用结果之后，就将子窗体变得可以滚动
- 甚至是在子窗体被删除而基窗体(strdcr)已经刷新之后，屏幕上的文本仍然保持原样
- 这是因为子窗体实际更新的是stdscr的字符数据

子窗体示例

```
#include <unistd.h>
#include <stdlib.h>
#include <curses.h>
#define NUM_NAMES 14
int main()
{
  WINDOW *sub_window_ptr;
  int x_loop;
  int y_loop;
  int counter;
  char a_letter = 'A';

  char *names[NUM_NAMES] = {"David Hudson,", "Andrew Crolla,",
    "James Jones,", "Ciara Loughran,", "Peter Bradley,",
    "Nancy Innocenzi,", "Charles Cooper,", "Rucha Nanavati,
    ", "Bob Vyas,", "Abdul Hussain,", "Anne Pawson,",
    "Alex Hopper,","Russell Thomas,", "Nazir Makandra,"};
```

子窗体示例

```
initscr();
for (y_loop = 0; y_loop < LINES - 1; y_loop++) {
  for (x_loop = 0; x_loop < COLS - 1; x_loop++) {
    mvwaddch(stdscr, y_loop, x_loop, a_letter);
    a_letter++;
    if (a_letter > 'Z') a_letter = 'A';
  }
}

sub_window_ptr = subwin(stdscr, 10, 20, 10, 10);
scrollok(sub_window_ptr, 1);
touchwin(stdscr);
refresh();
sleep(2);
```

子窗体示例

```
    werase(sub_window_ptr);
    mvwprintw(sub_window_ptr, 2, 0, "%s",
    "This window will now scroll as names are added ");
    wrefresh(sub_window_ptr);
    sleep(2);
    for (counter = 0; counter < NUM_NAMES; counter++) {
      wprintw(sub_window_ptr, "%s ", names[counter]);
      wrefresh(sub_window_ptr);
      sleep(2);
    }

    delwin(sub_window_ptr);
    touchwin(stdscr);
    refresh();
    sleep(2);
    getch();
    endwin();
    exit(EXIT_SUCCESS);
}
```

keypad模式

- 功能键的处理
  - 在大多数的终端上会发送一个以转义字符开始的字符串
  - 这些程序所具有的不仅是单击Escape键和由按下一个功能键所引起的字符串之间区别的问题，而且他必须使用相同逻辑按键的不同序列来处理不同的终端
  - curses提供了一个优雅的实用功能来管理这些功能按键
    - ✓ 对于每一个终端，每一个功能键所发送的字符序列都会被存储，通常是存储在一个terminfo结构中
    - ✓ 所包含的头文件curses.h具有一个以KEY_为前缀的定义部分定义了逻辑按键
  - 当curses启动时，序列与逻辑按键之间的转换就被禁止，必须使用keypad函数来打开
    - ✓ int keypad(WINDOW *window_ptr, bool keypad_on);
    - ✓ 如果函数调用成功则会返回OK，否则返回ERR

keypad模式的三个限制

- 转义序列的识别是时间相关的
  - 许多的网络协议会将字符组装到数所包中(会导致不能正确的识别转义序列)
  - 或者是分割他们(从而会导致功能按键序列会被识别为Escape与单个的字符)
- 唯一解决办法
  - 进行编程，使用信号来处理希望使用的每一个功能按键，为其发送单一的，唯一的字符
  - 为了使得curses可以区分按下Escape与以Escape开头的键盘序列，他必须等待一小段时间
  - 有时，一旦打开了keypad模式，Escape按键处理上的一个非常小的延时也会被注意到
- curses不能处理不唯一的转义序列
  - 如果终端有可以发送相同序列的两个不同的按键，curses只是简单的不处理这个序列，因为他不能确定应返回哪一个逻辑按键

keypad模式示例

❖ keypadtest.c

```
#include <curses.h>
#define MIN(a,b) a<b?a:b
#define MAX(a,b) a>b?a:b

int main(int argc, char *argv[]) {
  int x = 10;        /*待显示的字母坐标*/
  int y = 10;
  char ch = 'A';     /*待显示的字母*/

  initscr( );
  crmode( );         /*"中断模式"*/
  noecho( );         /*关闭回显*/
  clear( );
  keypad(stdscr, TRUE); /*打开keypad*/
  mvaddch(y, x, ch);
  chtype input;
  while ( (input=getch( )) && input!=ERR && input!='q' ) {
    if ( (input>='A' && input<='Z') ||
     (input>='a' && input<='z') )
      ch = input;
```

keypad模式示例

❖ keypadtest.c

```
    else{
      /*通过方向键调整坐标*/
      switch(input) {
      case KEY_LEFT:
      x = MAX(x-1, 0);
      break;
      case KEY_RIGHT:
      x = MIN(x+1, COLS);
      break;
      case KEY_UP:
      y = MAX(y-1, 0);
      break;
      case KEY_DOWN:
      y = MIN(y+1, LINES);
      break;
      }
    }
    clear( );
    mvaddch(y, x, ch);
    refresh( );
  }
  endwin( );
  return 0;
}
```

彩色显示

- 大多数早期的curses版本并不会支持颜色
- 颜色被 ncurses以及大多数现在的curses实现所支持
- curses中的颜色支持有一些不同，其原因在于每一个字符的颜色并不是独立于其底色而定义的
- 所以必须同时定义前景色与背景色，即所谓的颜色对

颜色检测

- **bool has_colors(void);**
  - 如果支持颜色，has_colors就会返回真
- **int start_color(void);**
  - 进行了颜色的初始化，如果颜色初始化成功，则返回OK
  - 初始化COLORS和COLOR_PAIR。
    - ✓ COLORS：终端所支持的最多的颜色数目
    - ✓ COLOR_PAIR：用户可以定义的色彩对的最大数目
- 系统颜色
  - COLOR_BLACK 0 黑色
  - COLOR_RED 1 红色
  - COLOR_GREEN 2 绿色
  - COLOR_YELLOW 3 黄色
  - COLOR_BLUE 4 蓝色
  - COLOR_MAGENTA 5 洋红色
  - COLOR_CYAN 6 蓝绿色, 青色
  - COLOR_WHITE 7 白色

初始化颜色对

- int init_pair(short pair_number, short foreground, short background);
  - 用于更改一个彩色对的定义
  - 彩色对是Curses的一个概念，它用一个整型数值去标志一对前景/背景彩色
  - pair_number: 彩色对数值，其范围从1到COLOR_PAIRS-1;
  - f: 指定前景彩色;
  - b: 指定背景彩色
- int COLOR_PAIR(int pair_number);
- int pair_content(short pair_number, short *foreground, short *background);

初始化颜色过程

- 定义颜色对1，使其背景色为绿色而前景色为红色
  - init_pair(1, COLOR_RED, COLOR_GREEN);
- 使用COLOR_PAIR将这个颜色作为一个属性来进行访问
  - wattron(window_ptr, COLOR_PAIR(1));
    - ✓ 将屏幕设置为绿色的背景色以及红色的前景色
- 说明
  - 因为COLOR_PAIR是一个属性，所以可以将其与其他的属性进行组合
  - 在PC上，经常可以通过使用位或操作符组合COLOR_PAIR属性与A_BOLD属性来获得屏幕的亮度
    - ✓ wattron(window_ptr, COLOR_PAIR(1) | A_BOLD);

颜色示例


tlmb@ridcully:~/chap06_curses
There are 8 COLORS, and 64 COLOR_PAIRS available
Color pair 5
Bold color pair 3
Bold color pair 5

颜色示例

```
int main()
{
    int i;

    initscr();

    if (!has_colors()) {
        endwin();
        fprintf(stderr, "Error - no color support on this terminal\n");
        exit(1);
    }

    if (start_color() != OK) {
        endwin();
        fprintf(stderr, "Error - could not initialize colors\n");
        exit(2);
    }
```

颜色示例

```
clear();
mvprintw(5, 5, "There are %d COLORS, and %d COLOR_PAIRS available",
         COLORS, COLOR_PAIRS);
refresh();
```

```
init_pair(1, COLOR_RED, COLOR_BLACK);
init_pair(2, COLOR_RED, COLOR_GREEN);
init_pair(3, COLOR_GREEN, COLOR_RED);
init_pair(4, COLOR_YELLOW, COLOR_BLUE);
init_pair(5, COLOR_BLACK, COLOR_WHITE);
init_pair(6, COLOR_MAGENTA, COLOR_BLUE);
init_pair(7, COLOR_CYAN, COLOR_WHITE);
```

颜色示例

```
    for (i = 1; i <= 7; i++) {
        attroff(A_BOLD);
        attrset(COLOR_PAIR(i));
        mvprintw(5 + i, 5, "Color pair %d", i);
        attrset(COLOR_PAIR(i) | A_BOLD);
        mvprintw(5 + i, 25, "Bold color pair %d", i);
        refresh();
        sleep(1);
    }

    endwin();
    exit(EXIT_SUCCESS);
}
```

重新定义颜色

- 在初始化颜色的时候改变某个颜色的RGB值
- init_color(COLOR_RED, 700, 0, 0);
  - 参数1：颜色名称
  - 参数2, 3, 4：分别为R(red),G(green),B(blue)的数值
    - ✓ 最小值：0
    - ✓ 最大值：1000
  - 如果显示终端无法改变颜色设置，函数将返回ERR。
- can_change_color()
  - 监测终端是否可以支持颜色改变

pad

- 背景
  - 期望得到一个比实际的物理屏幕要大的逻辑屏幕，并且每次只显示逻辑屏幕的部分内容
- 窗口缺陷
  - 所有窗体必须不大于物理屏幕
- pad功能
  - 操作并不适合普通窗体的逻辑屏幕信息
  - pad结构与WINDOW结构相类似
  - 所有可以用于向窗体输出的函数也可以用于pad
  - 但pad具有其特殊的创建与刷新例程

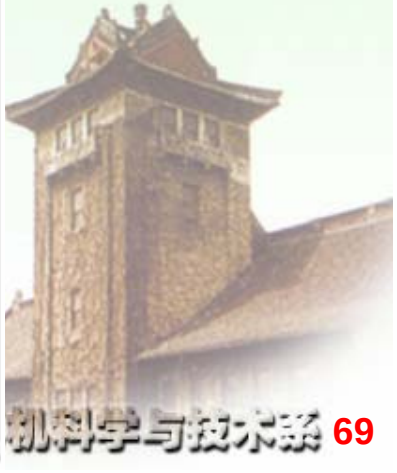
pad的创建与刷新

- WINDOW *newpad(int lines, int columns);
  - 返回为一个指向WINDOW结构的指针，与newwin函数相同。
  - 删除pad使用delwin函数，与窗体相同。
  - pad并没有限定一个特定的屏幕位置，必须指定希望pad出现在屏幕上的区域
- int prefresh(WINDOW *pad_ptr, int pad_row, int pad_column, int screen_row_min, int screen_col_min, int screen_row_max, int screen_col_max);
  - 创建一个pad区域，由(pad_row,pad_column)开始
  - 所定义的区域为(screen_row_min,screen_col_min)到(screen_row_max,screen_col_max)

pad示例

timb@ridcully:~/chap06_curses

```
yzabcdef
rstuvwxy
klmnopqr
deffghijklmnopqrst
wxyyzabcdefghijklm
pqrrstuvwxyzabcdef
ijkklmnopqrstuvwxy
bcddefghijklmnopqr
    uxyzabcdefghijk
    pqrstuvwxyzabcd
    ijklmnopqrstuvw
    bcdefghijklmnop
    uvwxyzabcdefghi
    nopqrstuvwxyzab
    ghijklmnopqrstu
    zabcdefghijklmn
    stuvwxyzabcdefg
    lmnopqrstuvwxyz
    efghijklmnopqrs
    xyzabcdefghijkl
```



pad示例

```
#include <unistd.h>
#include <stdlib.h>
#include <curses.h>
int main()
{
    WINDOW *pad_ptr;
    int x, y;
    int pad_lines;
int pad_cols;
char disp_char;
initscr();
pad_lines = LINES + 50;
pad_cols = COLS + 50;
pad_ptr = newpad(pad_lines, pad_cols);
disp_char = 'a';
for (x = 0; x < pad_lines; x++) {
    for (y = 0; y < pad_cols; y++) {
        mvwaddch(pad_ptr, x, y, disp_char);
        if (disp_char == 'z') disp_char = 'a';
        else disp_char++;
    }
}
```

pad示例

```
  prefresh(pad_ptr, 5, 7, 2, 2, 9, 9);
  sleep(1);
  prefresh(pad_ptr, LINES + 5, COLS + 7, 5, 5, 21, 19);
  sleep(1);
  delwin(pad_ptr);
  endwin();
  exit(EXIT_SUCCESS);
}
```

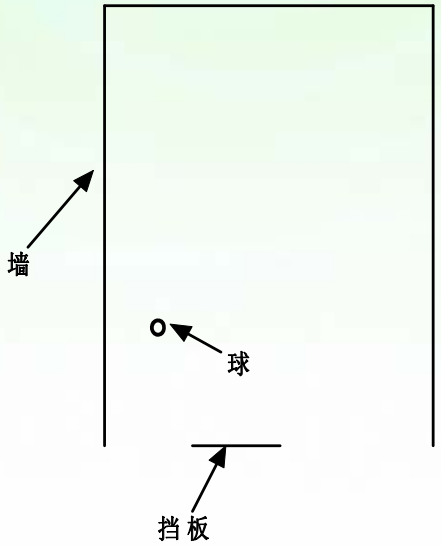
主要内容

- 背景知识
  - curses库概述
  - curses库编程
- 实验内容
  - 利用curses库实现弹球游戏
  - 利用多线程实现弹球游戏

利用curses库实现弹球游戏

❖实验说明

- 弹球游戏的界面如下图所示，由墙、球和挡板组成
- 游戏的主要规则为
  - ✓球以一定的速度移动
  - ✓球碰到墙壁或挡板会被弹回
  - ✓用户通过方向键来控制挡板左右移动



利用curses库实现弹球游戏

- 解决方案：需求分析
  - 运动轨迹
    - ✓ 球在没有碰到挡板或者墙的时候，会朝着一条直线一直运动
    - ✓ 当碰到挡板或者墙的时候，球运动的方向会改变，这涉及如何使用curses库实现动画效果。
  - 动画实现方法
    - ✓ 在一个地方画一个字符串，等待几毫秒，然后擦去旧影像并在原来位置的边上重新绘制一个相同字符串，通过这样的一个过程就能够形成动画效果。
    - ✓ 可以通过usleep( )函数让进程进入等待，当进程被唤醒时，绘制下一个图像，动画效果便能形成。
  - 场景球表示
    - ✓ 用字母O表示球
    - ✓ 球会先水平向右移动，当移动到屏幕最右端，球会水平向左移动
    - ✓ 当球碰到屏幕的左端，球会改变方向向右移动

利用curses库实现弹球游戏

❖解决方案： usleep()实现运动轨迹代码示例

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

int main(int argc, char *argv[]){
   int x = 10;
   int y = 10;
   int direction = 1;
   char ball = 'O';
   /*初始化*/
   initscr( );
   crmode( );
   noecho( );
```

利用curses库实现弹球游戏

❖解决方案： usleep()实现运动轨迹代码示例

```
while ( true ) {
   clear( );
   mvaddch(y, x, ball);
   refresh( );
   /*更新坐标*/
   x += direction;
   /*变换方向*/
   if ( x==COLS ) {
      direction = -1;
      x = COLS-1;
      beep( ); /*碰到墙时，发出声音*/
   }
   if ( x<0 ){
      direction = 1;
      x = 0;
      beep( );
   }
   usleep(100000);/*睡眠*/
}
endwin( );
return0;
}
```

利用curses库实现弹球游戏

❖解决方案：基于定时器实现运动轨迹代码示例

```
#include <curses.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>
/*球的坐标
int x = 10;
int y = 10;
int direction = 1;
char ball = 'O';
/*定时器设置函数*/
int set_ticker(long n_msecs ) {
  struct itimerval new_timeset;
  long     n_sec, n_usecs;
  n_sec = n_msecs / 1000 ;
  n_usecs = ( n_msecs % 1000 ) * 1000L ;
  new_timeset.it_interval.tv_sec  = n_sec;
  new_timeset.it_interval.tv_usec = n_usecs;
  new_timeset.it_value.tv_sec     = n_sec;
  new_timeset.it_value.tv_usec    = n_usecs ;
  return setitimer(ITIMER_REAL, &new_timeset, NULL);
}
```

利用curses库实现弹球游戏

❖解决方案：基于定时器实现运动轨迹代码示例

```
/*信号处理函数，在SIGALRM信号产生时，绘制图像*/
void paint( ){
   clear( );
   mvaddch(y, x, ball);
   refresh( );

   x += direction;
   if ( x==COLS ) {
      direction = -1;
      x = COLS-1;
      beep( );
   }
   if ( x<0 ){
      direction = 1;
      x = 0;
      beep( );
   }
}
```

利用curses库实现弹球游戏

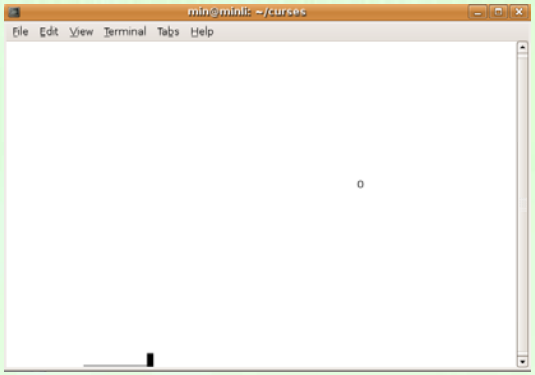
❖解决方案：基于定时器实现运动轨迹代码示例

```
int main(int argc, char *argv[]){
  chtype input;
  long delay = 100;
    /*初始化curses*/
    ……
    /*设定定时器*/
  signal(SIGALRM, paint);
  set_ticker( delay );
  while ( (input=getch( )) &&
    input!=ERR && input!='q' ) {
    switch( input ){
      case 'f':{
        /*加速*/
        delay /= 2;
        set_ticker( delay );
        break;
      }
      case 's':{
        /*减速*/
        delay *= 2;
        set_ticker( delay );
        break;
      }
    }
  }
  endwin( );
  return 0;
}
```

利用curses库实现弹球游戏

❖解决方案：绘制球与挡板

- 在弹球游戏中，球的运动是自动的，球根据碰到的墙或挡板自动进行方向变换。
- 而挡板的移动是由游戏者控制的，游戏者通过方向键控制挡板的左右移动。

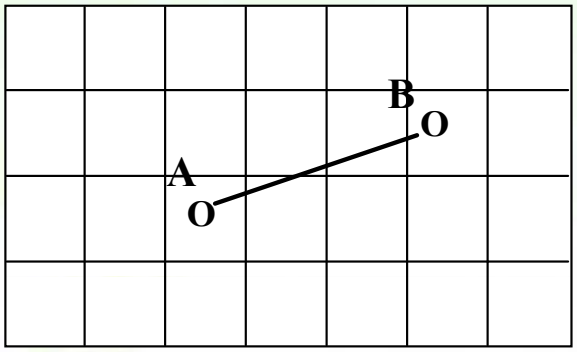


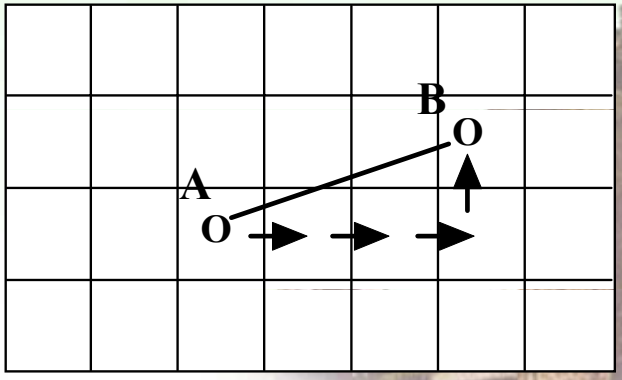

```
#define MAX(a,b)  a>b?a:b;
#define MIN(a,b)  a<b?a:b;

int ballx = 10;
int bally = 10;
int direction = 1;
char ball = 'o';
int barx = 10;
int bary ;
char* bar="__________";
int barlength = 10;
```

```
case KEY_RIGHT:{
  barx = MIN(barx+1, COLS-1-barlength);
  break;
}
case KEY_LEFT:{
  barx = MAX( barx-1, 0);
  break;
}
```

利用curses库实现弹球游戏

- 解决方案：斜线运动
  - 弹球游戏中，球以斜线方式运动。在绘制斜线运动的动画时，需要考虑运动的平滑性。在下左图中，从A往B方向移动。
    - ✓ 如果A先向上移动一个单位，则必须同时向右移动3个单位。这样的移动跨度较大，移动不平滑，动画显示时有跳跃感。
    - ✓ 如果要平滑移动，比较好的方式是按照右图的移动方式，每次移动一个单位。
    - ✓ 当球遇到挡板或墙时，球运动的斜率也要进行相应的变化，该段代码框架如下并可由用户自行实现。





主要内容

- 背景知识
  - curses库概述
  - curses库编程
- 实验内容
  - 利用curses库实现弹球游戏
  - 利用多线程实现弹球游戏

利用多线程实现弹球游戏

❖实验说明

- 前一个实验中，通过定时器实现动画的绘制
  - ✓程序通过不断响应SIGALRM信号，定时对屏幕进行刷新
  - ✓这该方案实际上是对多线程的一种模拟，通过信号机制，程序可以同时进行用户输入响应和屏幕绘制
- 在本实验中，要求通过多线程实现动画的绘制

❖解决方案

- 主程序进行用户输入的响应，程序创建出一个独立的线程进行屏幕的绘制
- 屏幕绘制进程可以通过usleep函数进行休眠，完成屏幕的定时重绘

利用多线程实现弹球游戏

❖解决方案：多线程处理代码框架

```
void* paintThread(void* arg);
int main( ) {
        ……
        pthread_create(…..)/*创建屏幕绘制线程*/
……
}
void* paintThread(void* arg){
   while (true) {
                /*绘制屏幕*/
                /*通过usleep()休眠*/
        }
}
```

第10章 事件驱动编程