Программирование с помощью ncurses, часть вторая

Андрей Боровский

Мы продолжаем знакомство с ncurses. В прошлый раз мы научились создавать окна. На этом уроке мы рассмотрим другие важные возможности ncurses, такие, как управление цветом и поддержка мыши.

Управление цветом

Принципы работы с цветом в ncurses могут оказаться неожиданными для тех, кто привык работать с цветами в растровых графических системах (и для тех, кто имеет опыт работы с текстовым режимом DOS/Windows). Библиотека ncurses инициализирует восемь базовых цветов: черный, красный, зеленый, желтый, синий (blue), ярко-красный (magenta), голубой (cyan) и белый (базовыми называются цвета с обычным уровнем яркости). Поскольку к каждому базовому цвету можно применить атрибут повышенной яркости A_BOLD, мы получаем всего 16 цветов (в результате применения атрибута A_BOLD к черному цвету получается темно-серый цвет). Базовым цветам соответствуют константы COLOR_BLACK, COLOR_RED, COLOR_GREEN, COLOR_YELLLOW, COLOR_BLUE, COLOR_MAGENTA, COLOR_CYAN и COLOR_WHITE (для черного, красного, зеленого, желтого, синего, ярко-красного, голубого и белого цветов соответственно). Следует отметить, что фактические цвета в окне терминала зависят, прежде всего, от настроек самого терминала. Например, базовый желтый цвет (COLOR_YELLLOW) будет выглядеть скорее как коричневый, а для того, чтобы он стал, собственно, желтым, ему необходимо придать атрибут повышенной яркости. Библиотека ncursese позволяет определять собственные цвета с помощью функции init_color, но эта возможность поддерживается не всеми консолями. Позволяет ли консоль определять собственные цвета, можно выяснить с помощью функции can_change_color(). Цвета в ncurses объединяются в пары – цвет символов (foreground) и цвет фона (background). Перед тем как печатать цветной текст, необходимо определить соответствующую цветовую пару и установить ее в качестве атрибута текста (так же как устанавливается атрибут мигания или подчеркивания). Фактически номер цветовой пары является одним из атрибутов символа. Изменить цвет фона или цвет символов независимо друг от друга нельзя, необходимо определять новую пару.

Система управления цветами ncurses инициализирует две переменные – COLORS (количество базовых цветов) и COLOR_PAIRS (максимальное количество цветовых пар, которые можно определить одновременно). При работе с терминалом konsole эти переменные принимают значения 8 и 64 соответственно.

Рассмотрим управление цветом на примере программы cursedcolors (на диске – файл cursedcolors.c)

```
#include <termios.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <stdlib.h>
#include <curses.h>

void sig_winch(int signo)
{
    struct winsize size;
    ioctl(fileno(stdout), TIOCGWINSZ, (char *) &size);
```

```
    resizeterm(size.ws_row, size.ws_col);
}

int main(int argc, char ** argv)
{
    WINDOW * wnd;
    WINDOW * subwnd;

    initscr();
    signal(SIGWINCH, sig_winch);
    curs_set(FALSE);
    start_color();
    refresh();
    init_pair(1, COLOR_BLUE, COLOR_GREEN);
    init_pair(2, COLOR_YELLOW, COLOR_BLUE);
    wnd = newwin(5, 18, 2, 4);
    wattron(wnd, COLOR_PAIR(1));
    box(wnd, '|', '-');
    subwnd = derwin(wnd, 3, 16, 1, 1);
    wbkgd(subwnd, COLOR_PAIR(2));
    wattron(subwnd, A_BOLD);
    wprintw(subwnd, "Hello, brave new curses world!\n");
    wrefresh(subwnd);
    wrefresh(wnd);
    delwin(subwnd);
    delwin(wnd);
    wmove(stdscr, 8, 1);
    printw("Press any key to continue...");
    refresh();
    getch();
    endwin();
    exit(EXIT_SUCCESS);
}
```

Эта программа основана на программе cursedwindows из предыдущей статьи, так что многие ее части должны быть вам знакомы. Функция start_color() инициализирует управление цветом ncurses. Остальные функции, связанные с цветом, можно вызывать только после вызова start_color(). Новые цветовые пары создаются с помощью функции init_pair(). Первым параметром init_pair() должен быть один из допустимых номеров пары (от 1 до COLOR_PAIRS-1). Вторым параметром функции init_pair() должна быть константа, обозначающая базовый цвет символа, а третьим – константа, обозначающая базовый цвет фона. Цветовая пара с номером 0 определена в ncurses как «белый на черном» и изменить ее нельзя. Мы создаем две пары цветов – «синие символы на зеленом фоне» под номером 1, и «желтые символы на синем фоне» (любимое сочетание цветов небезызвестного Питера Нортона), под номером 2. Номер цветовой пары служит ее идентификатором. Для того чтобы сделать выбранную цветовую пару атрибутом выводимого текста, необходимо установить с помощью функции attron/wattron атрибут COLOR_PAIR(X), где X – номер цветовой пары. Атрибут COLOR_PAIR(X) можно комбинировать с другими атрибутами, например, с атрибутом A_BOLD, который влияет на яркость цвета символов (но не на яркость цвета фона). Для того чтобы изменить яркость фона, придется комбинировать этот атрибут с атрибутом A_REVERSE.

Вызов функции

```
wattron(wnd, COLOR_PAIR(1));
```

устанавливает цвет фона и символов (цветовую пару 1) для «внешнего» окна wnd, содержащего рамку. Теперь функция box() напечатает символы рамки с учетом заданных атрибутов цвета. Функция wbkgd() позволяет нам заполнить

структуру данных, соответствующую массиву символов окна, заданными атрибутами текста. Вызов

```
wbkgd(subwnd, COLOR_PAIR(2));
```

заполняет окно subwnd фоновым цветом из цветовой пары 2 и устанавливает соответствующий цвет символов в окне. Помимо атрибута COLOR_PAIR() этой функции можно передавать все те же комбинации атрибутов, что и wattron(). Атрибуты затем будут применены к тексту, выводимому в окне по умолчанию. Для того чтобы сделать цвет шрифта в окне subwnd ярким, мы вызываем функцию wattron() с атрибутом A_BOLD. Заметьте, что в функции wattron(), вызванной для окна subwnd, мы не указываем цветовую пару, поскольку в этом нет необходимости. Функция wbkgd() уже заполнила символьный массив окна subwnd нужными атрибутами цвета и нам остается только указать атрибут яркости. В принципе, мы могли бы обойтись и без вызова wattron(), если бы вызов wbkgd() выглядел так:

```
wbkgd(subwnd, COLOR_PAIR(2)|A_BOLD);
```

Теперь мы можем распечатать текст в окне с помощью функции wprintw(). Для того чтобы символы, напечатанные в окне, стали видимыми, мы должны, как всегда, вызвать функцию wrefresh(). Теперь окно с текстом и обрамляющая его рамка сияют разными цветами (рис. 1). Вы могли заметить, что если в обычном режиме окно терминала было, например, белым, то во время работы программы cursedcolors оно становится черным. Это происходит потому, что по умолчанию при инициализации цвета окно stdscr заполняется атрибутами цветовой пары 0, как если бы была вызвана функция

```
wbkgd(stdscr, COLOR_PAIR(0));
```

В результате надпись “Press any key to continue...”, которую мы печатаем в окне stdscr, выводится белым шрифтом на черном фоне.

Рисунок 1. Разноцветные окна в ncurses

При работе с цветом в ncurses следует помнить о том, что массивы данных окон хранят только номера цветовых пар, применяемых к каждой ячейке, а не сами значения цветов. Из этого следует, что цвета уже напечатанного текста зависят от определения цветовых пар. Допустим, вы определили цветовую пару 1 как «желтый на синем» и напечатали какой-нибудь текст, используя эту пару в качестве атрибута. Если затем вы переопределите цветовую пару 1 как «красный на белом», цвета шрифта и фона в уже напечатанном тексте изменятся соответственно новому определению цветовой пары.

Ввод данных в окнах

С одной из функций ввода данных – getch(), мы уже познакомились. Мы знаем так же, что этой функции соответствует «оконная» функция wgetch(). С помощью этих функций вы можете считывать с экрана отдельные символы. В отличие от них, семейство функций getstr()/getnstr/wgetstr()/wgetnstr() позволяет считывать целые строки. Буква n перед str в именах функций свидетельствует о том, что эти варианты функций позволяют указать максимальную длину строки-буфера и, тем самым, избежать его переполнения при вводе. Программа cursedinput (ее исходные тексты вы найдете в файле cursedinput.c) позволяет пользователю вводить строку текста и, затем, распечатывает введенную строку.

```
#include <termios.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <stdlib.h>
#include <curses.h>

#define MAX_NAME_LEN 15

void sig_winch(int signo)
{
    struct winsize size;
    ioctl(fileno(stdout), TIOCGWINSZ, (char *) &size);
    resizeterm(size.ws_row, size.ws_col);
}

int main(int argc, char ** argv)
{
    WINDOW * wnd;
    char name[MAX_NAME_LEN + 1];
    initscr();
    signal(SIGWINCH, sig_winch);
    curs_set(TRUE);
    start_color();
    refresh();
    init_pair(1, COLOR_YELLOW, COLOR_BLUE);
    wnd = newwin(5, 23, 2, 2);
    wbkgd(wnd, COLOR_PAIR(1));
    wattron(wnd, A_BOLD);
    wprintw(wnd, "Enter your name...\n");
    wgetnstr(wnd, name, MAX_NAME_LEN);
    name[MAX_NAME_LEN] = 0;
    wprintw(wnd, "Hello, %s!", name);
    wrefresh(wnd);
    delwin(wnd);
    curs_set(FALSE);
    move(8, 4);
    printw("Press any key to continue...");
    refresh();
    getch();
    endwin();
    exit(EXIT_SUCCESS);
}
```

Поскольку в программе curedinput пользователь должен вводить данные, нам удобно сделать курсор видимым (что мы и делаем с помощью вызова curs_set(TRUE)). Собственно ввод строки выполняется с помощью wgetnstr(). Первый параметр функции – идентификатор окна, в котором вводятся данные (то есть отображаются вводимые символы и курсор), второй параметр – строка-буфер, в которую записываются введенные символы, а третьим параметром является длина буфера. При работе с программой вы увидите, что

нельзя ввести число символов, превышающее MAX_NAME_LEN. Перед выводом строки "Press any key to continue..." мы снова прячем курсор.

Режим работы терминала, в котором была запущена программа (программа наследует режим работы терминала от программы, которая ее запустила), может повлиять на поведение некоторых функций ввода данных. В одной из предыдущих статей мы уже упоминали о каноническом и неканоническом режиме работы терминала. В каноническом режиме терминал буферизует вводимые данные и передает их программе только после того, как пользователь нажмет [Enter]. В неканоническом режиме вводимые символы передаются программе немедленно. Режим работы терминала можно изменить с помощью функций cbreak() и nocbreak(). В результате вызова cbreak() терминал переходит в режим, в котором введенные символы предаются программе, не дожидаясь нажатия [Enter], а клавиша [BackSpace] игнорируется. Терминал выводится из режима cbreak() с помощью вызова функции nocbreak().

Работая с программой cursedinput, вы, конечно, заметили, что функция wgetnstr() допускает редактирование вводимой строки с помощью клавиши BackSpace. Поведение этой функции не зависит от режима cbreak()/nocbreak(), но поведение других функций, в частности, getch(), - зависит. В режиме nocbreak() getch() возвращает управление программе только после того, как пользователь нажмет [Enter]. Все наши программы (как и большинство программ ncurses) устанавливают режим cbreak().

Функция для ввода пароля

Чтобы лучше изучить возможности ввода текста в библиотеке ncurses, напишем функцию, предназначенную для ввода пароля. Вместо вводимых символов наша функция печатает на экране звездочки (рис. 2) и позволяет редактировать строку с помощью клавиши [BackSpace]. Мы перепишем программу cursedinput так, чтобы вместо своего имени пользователь вводил пароль. Этот пароль затем сверяется с константой, заданной в программе, и в зависимости от того, совпадает ли пароль с константой или нет, выводится сообщение о предоставлении доступа или отказе в нем. Исходный текст новой программы вы найдете в файле cusedpassword.c. Мы приводим только фрагмент, который претерпел изменения по сравнению с программой cursedinput.

```
keypad(wnd, TRUE);
wprintw(wnd, "Enter password...\n");
get_password(wnd, password, MAX_LEN);
wattron(wnd, A_BLINK);
if (strcmp(password, RIGHT_PASSWORD) == 0)
  wprintw(wnd, "ACCESS GRANTED!");
else
  wprintw(wnd, "ACCESS DENIED!");
```

У определенной нами функции get_password() три параметра – идентификатор окна, в котором выполняется ввод, адрес буфера, в который записываются вводимые символы и число, указывающее длину буфера (вместе с завершающим нулем). Прототип функции strcmp(), которую мы используем для сравнения переданной пользователем строки и пароля, находится в файле <string.h>. Рассмотрим теперь саму функцию get_password():

```
void get_password(WINDOW * win, char * password, int max_len)
{
    int i = 0;
    int ch;
    while (((ch = wgetch(win)) != 10) && (i < max_len-1)) {
      if (ch == KEY_BACKSPACE) {
```

```
            int x, y;
            if (i==0) continue;
            getyx(win, y, x);
            mvwaddch(win, y, x-1, ' ');
            wrefresh(win);
            wmove(win, y, x-1);
            i--;
            continue;
          }
          password[i++] = ch;
          wechochar(win, '*');
        }
        password[i] = 0;
        wechochar(win, '\n');
}
```

Функция get_password() считывает символы из входного потока с помощью функции wgetch() до тех пор, пока пользователь не нажмет ввод, или пока длина введенной строки не сравняется с максимально допустимой длиной. Для вывода отдельных символов в окно можно применить функцию waddch(), однако мы используем функцию wechochar(), которая эквивалента вызову waddch() с последующим вызовом wrefresh(). Самая сложная часть функции get_password() связана с обработкой нажатия клавиши [BackSpace]. Прежде всего, необходимо получить код этой специальной клавиши. По умолчанию при нажатии на специальные клавиши, такие как стрелки, клавиши [F1]-[F12] или [BackSpace], терминал генерирует последовательность кодов, представляющих собой так называемую Esc-последовательность. Для того чтобы заменить Esc-последовательность одним специальным кодом, необходимо вызвать функцию keypad() с ненулевым вторым параметром (что мы и делаем в главной функции программы). Первым параметром keypad() должен быть идентификатор окна (в нашем случае – wnd).

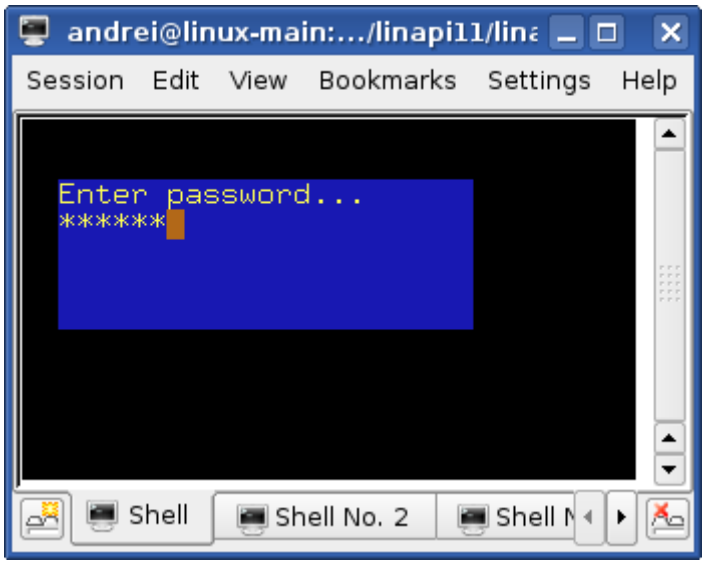

Рисунок 2. Ввод пароля с помощью функции get_password()

Вызов keypad() с ненулевым вторым параметром приводит к тому, что клавиши [F1]-[F12] генерируют коды KEY_F1-KEY_F12, клавиши со стрелками – коды KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT, а клавиша [BackSpace] – код KEY_BACKSPACE (описание других кодов специальных клавиш и событий вы найдете на странице man функции getch()). Получив в потоке ввода код KEY_BACKSPACE, мы должны выполнить несколько операций. Прежде всего, мы стираем только что напечатанную звездочку. Для этого нужно получить текущие координаты курсора, сдвинуть курсор на одну позицию влево и

напечатать пробел. Затем курсор снова нужно сдвинуть на одну позицию влево. Получить текущие координаты курсора в окне можно с помощью макроса getyx(). Первым параметром макроса является идентификатор окна, вторым параметром – переменная, в которой макрос сохранит значение строки курсора, третьим параметром – переменная, в которой будет сохранено значение столбца курсора. Именно потому, что getyx() – макрос, мы передаем для получения значений переменные, а не указатели на них. Функция mvwaddch() сочетает перемещение курсора и вывод символа. Первый параметр функции – идентификатор окна. За ним следуют новые координаты курсора – строка и столбец. Последним параметром функции является символ, который нужно напечатать. После того, как мы привели в порядок экран, мы уменьшаем на единицу счетчик введенных символов (переменная i). Если переменная i равна нулю, никаких действий не выполняется. Наша функция get_password() будет работать правильно только в режиме cbreak(). Следует отметить также, что функция не будет работать корректно с клавишей [BackSpace], если при вводе пароля произошел перенос строки.

Окна и мыши

Еще одной важной возможностью, которую ncurses предоставляет программистам, является поддержка мыши в окне терминала. Рассмотрим программу cursedmouse (на диске – файл cursedmouse.c), которая регистрирует щелчки левой кнопкой мыши, сделанные пользователем в окне терминала, и распечатывает координаты курсора мыши в момент щелчка. Ради простоты мы не создаем в этой программе никаких окон (кроме окна stdscr, которое создается автоматически).

```
#include <termios.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <stdlib.h>
#include <curses.h>

void sig_winch(int signo)
{
    struct winsize size;
    ioctl(fileno(stdout), TIOCGWINSZ, (char *) &size);
    resizeterm(size.ws_row, size.ws_col);
    nodelay(stdscr, 1);
    while (wgetch(stdscr) != ERR);
    nodelay(stdscr, 0);
}

int main(int argc, char ** argv)
{
    initscr();
    signal(SIGWINCH, sig_winch);
    keypad(stdscr, 1);
    mousemask(BUTTON1_CLICKED, NULL);
    move(2,2);
    printw("Press the left mouse button to test mouse\n");
    printw("Press any key to quit...\n");
    refresh();
    while (wgetch(stdscr) == KEY_MOUSE) {
       MEVENT event;
       getmouse(&event);
       move(0, 0);
       printw("Mouse button pressed at %i, %i\n", event.x, event.y);
       refresh();
       move(event.y, event.x);
    }
```

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

Поддержка мыши в ncurses инициализируется с помощью функции mousemask(). Первым параметром этой функции должна быть маска событий мыши, которые следует обрабатывать в программе, вторым параметром может быть указатель на переменную, в которой функция сохранит прежнюю маску событий или NULL, если прежняя маска нам не нужна. Каждому событию мыши в ncurses соответствует константа. Если мы хотим обрабатывать несколько событий мыши, при вызове функции mousemask() мы должны объединить соответствующие константы операцией «|». Повторный вызов mousemask() приведет к установке новой маски событий (вызов mousemask() с первым аргументом, равным 0, отключает поддержку мыши).

Рассмотрим некоторые константы, определяющие события мыши. Константа BUTTON1_CLICKED соответствует щелчку левой кнопкой мыши (точнее говоря, - щелчку первой кнопкой; будет ли первая кнопка левой кнопкой мыши, зависит от настроек мыши). Константа BUTTON2_PRESSED указывает, что программа должна реагировать на нажатие пользователем второй (обычно – правой) кнопки мыши. Константа REPORT_MOUSE_POSITION указывает, что мы хотим отслеживать движение указателя мыши, а константа ALL_MOUSE_EVENTS заставляет программу реагировать на все события мыши (более полное описание констант событий вы найдете на странице man функции mousemask(3x)). В качестве результирующего значения функция mousemask() возвращает маску из выбранных нами событий, которые фактически могут быть обработаны. Если функция возвращает 0, значит работа с мышью в консоли не поддерживается.

Каждый раз, когда в системе происходит одно из «наблюдаемых» событий мыши, в потоке ввода программы появляется специальный символ KEY_MOUSE. Точнее говоря, по умолчанию, в потоке ввода программы Linux появляется Esc-последовательность, соответствующая этому символу, так что в программе cursedmouse мы тоже должны вызвать функцию keypad() с ненулевым вторым параметром.

После того, как мы считали из потока ввода специальный символ KEY_MOUSE, мы можем получить более подробную информацию о вызвавшем его событии мыши. Делается это с помощью функции getmouse(). Аргументом функции getmouse() должен быть указатель на структуру MEVENT. Определение структуры MEVENT выглядит следующим образом:

```
typedef struct {
short id;         /* идентификатор для различения нескольких устройств */
int x, y, z;      /* координаты указателя в момент событи */
mmask_t bstate;   /* маска событий */
} MEVENT;
```

Координаты указателя возвращаются в формате строка (y), столбец (x). Поле bstate содержит один единственный бит, соответствующий константе события.

В программе cursedmouse мы считываем поступающие во входной поток символы в цикле. Если во входном потоке появляется символ KEY_MOUSE, мы, с помощью функции getmouse(), определяем координаты указателя мыши в момент события и распечатываем их (мы распечатываем строку с координатами в левом верхнем углу экрана, а затем переводим курсор туда, куда указывала мышь в момент события). Появление в потоке ввода символа, отличного от KEY_MOUSE, приводит к завершению программы.

Осталось обратить внимание читателя на обработку сигнала SIGWINCH в программе cursedmouse. Изменение размеров экрана при включенной

поддержке мыши приведет к появлению в потоке ввода символов Esc-последовательности специального символа KEY_RESIZE (это еще один способ предупредить программу о том, что размеры экрана изменились). В программе cursedmouse появление в потоке ввода каких-либо кодов, отличных от KEY_MOUSE, вызывает завершение программы. Для того чтобы избежать досрочного завершения, в обработчике сигнала SIGWINCH мы опустошаем поток ввода с помощью функции flushinp(). Естественно, этот способ спасения программы от досрочного завершения годится не всегда (ведь в момент изменения размеров окна терминала поток ввода может содержать важную информацию). Все это лишний раз демонстрирует, насколько нетривиальной является обработка изменения размеров экрана в программах ncurses.

На этом я заканчиваю (честное слово!) серию статей, посвященную Unix API. Я благодарю вас за внимание, проявленное к этой серии, и надеюсь, что с помощью моих статей вы получили некоторое общее представление о низкоуровневом программировании в Linux/Unix, а, самое главное, смогли ответить на вопрос, - нужно ли вам все это.