

## Menús con Curses

# Menú del Día

El tema de los menús con *curses* es complejo. La información que se puede consultar por ahí, es contradictoria. Una función puede aparecer como `menu_post()`, `display_menu()` o, incluso, `menu_display()` en según que versión del manual mires, cuando en realidad se llama `post_menu()`. Los ejemplos que se encuentran por Internet no están documentados y muchos de ellos no compilan. **POR PAUL C. BROWN**



¿Qué hacer? Acudir a la sección de interfaces del tío Paul, por supuesto. De hecho, para empezar a comprender como funcionan los menús, ni siquiera vamos a verlos con una envoltura C++, sino que vamos a estudiar unos sencillos programas en C pelado y mondado para evitar líos y confusiones.

Con tal motivo presento la prueba A: el listado 1, que contiene un programa, y sin que sirva de precedente, completo, compilable (se puede utilizar g++ o gcc, monta tanto, tanto monta) y ejecutable que presentará en pantalla nuestro primer menú (ver Figura 1)... Sí, ya sé que no es muy

impresionante, pero paciencia. Analicemos lo que hace el código y podremos mejorarlo.

## Primer menú

Lo primero es declarar una matriz de cadenas (*options*) que contiene las opciones que vamos a mostrar en nuestro menú. Atención, que si bien se mostrarán cuatro elementos en el menú, hay que definir 5 porque *curses* exige que el menú termine en una cadena *nula*, que es exactamente lo que es el elemento (*char\**) *NULL* al final de la matriz.

A continuación, nos metemos directamente en la función *main* del programa y inicializamos la ventana estándar de

*curses* (esto a estas alturas no debería entrañar ningún misterio).

De la línea 18 a la 21, declaramos variables que nos servirán en la creación del menú. Tenemos, por un lado, un puntero a una variable *MENU*, que es la que apunta al menú propiamente dicho. A continuación tenemos un puntero a las cadenas *options*, un entero *i* que servirá de iterador en un bucle *y*, finalmente, una matriz de punteros *ITEM* (*barra*) cada elemento del cual contendrá las opciones disponibles en el menú.

El siguiente paso consiste en asignar espacio de memoria a *barra*. Esto lo hacemos con la función C *calloc()* y le asignamos tanto espacio como elemen-

## Listado 1: menu.c - nuestro primer menú

```

01 #include <stdlib.h>
02 #include <menu.h>
03
04 char* opciones[5] =
05 {
06     "Fichero", "Editar",
07     "Ventana", "Ayuda", (char*)
08     NULL
09 };
10
11 int main(int argc, char
12     *argv[])
13 {
14     (void) initscr();
15     keypad(stdscr, TRUE);
16     (void) nonl();
17     (void) cbreak();
18     (void) noecho();
19     wclear(stdscr);
20
21     MENU * menu_principal;
22     char ** o=opciones;
23     int i;
24     ITEM ** barra;
25
26     barra=(ITEM**)calloc(5,sizeof(
27     ITEM*));
28
29     for(i=0;i<5;i++)
30         barra[i]=new_item(opciones[i],
31         "");
32
33     menu_principal=new_menu((ITEM*
34     *)barra);
35
36     post_menu(menu_principal);
37     refresh();
38
39     int ch;
40
41     while
42         ((ch=getch())!=KEY_HOME)
43     {
44         /* procesar entrada del
45         teclado */
46     }
47
48     unpost_menu(menu_principal);
49     refresh();
50     free_menu(menu_principal);
51
52     while(*barra)
53         free_item(*barra++);
54
55     endwin();
56     exit(0);
57 }

```

tos del menú disponemos (en este caso, 5). A continuación, en la línea 26, tenemos un bucle que asigna los elementos del menú a *barra* y, después, creamos el menú propiamente dicho pasando *barra* como parámetro a la función *new\_menu()* de *curses*.

Y, por fin, podremos ver nuestro menú. Lo mandamos a la ventana con *post\_menu()*. Como no se especifica ninguna otra ventana, el menú aparecerá en la ventana estándar por defecto, *stdscr*. Para que se pueda visualizar el menú, habremos de refrescar la pantalla, cosa que hacemos con *refresh()*.

El resto del listado debería ser obvio: se espera un entrada del teclado, si el usuario pulsa la tecla [Inicio], el bucle termina, se retira el menú, se liberan los elementos, se cierra *curses* y se acaba el programa. ¿Alguien está viendo maneras de envolver todas estas feas tareas domésticas en una hermética clase C++? Por que yo sí.

El ejemplo anterior es un poco aburrido ¿verdad? Pero si de verdad queréis probarlo, para compilar el listado 1, bastará con ejecutar...

```
gcc -o menu menu.c -lcurses
-lmenu
```

... en el directorio donde tengamos el código fuente. O, si preferís...

```
g++ -o menu menu.c -lcurses
-lmenu
```

... que lo compilará igual.

El aspecto de nuestro menú no es exactamente como lo queremos. Esto es porque no hemos definido la *geometría* del menú y la librería *menu* aplica la geometría por defecto, que, según la documentación, consiste en un menú de una columna y dieciséis filas. Además, no podemos desplazarnos por las opciones, lo cual hace que nuestro menú sea incluso menos interesante.

## Listado 2: Menú mejorado.

```

01 .
02 .
03 .
04
05     menu_principal=new_menu((ITEM*
06     *)barra);
07
08     set_menu_format(menu_principal
09     ,1,5);
10
11     post_menu(menu_principal);
12     refresh();
13
14     int ch;
15
16     while ((ch=getch())!=KEY_HOME)
17     {
18         switch(ch)
19         {
20             case KEY_RIGHT:
21
22                 menu_driver(menu_principal,
23                 REQ_RIGHT_ITEM);
24                 break;
25             case KEY_LEFT:
26
27                 menu_driver(menu_principal,
28                 REQ_LEFT_ITEM);
29                 break;
30             default:
31
32                 menu_driver(menu_principal,
33                 ch);
34                 break;
35         }
36     }
37
38 .
39 .
40 .

```

Vamos a ver como se solucionan los dos problemas anteriores y después pasaremos a abstraer el código a una clase.

## Un poco de Movimiento

En el listado 2 podemos ver una versión mejorada del programa, o al menos la parte que nos atañe. Observemos, por ejemplo, que hemos insertado una nueva función, *set\_menu\_format()*, a la cual, pasándole el puntero del menú el cual queremos modificar y las filas y las columnas, modifica el formato por defecto. Como nosotros queremos tener una barra de menús, utilizamos una sola fila y varias columnas, tantas

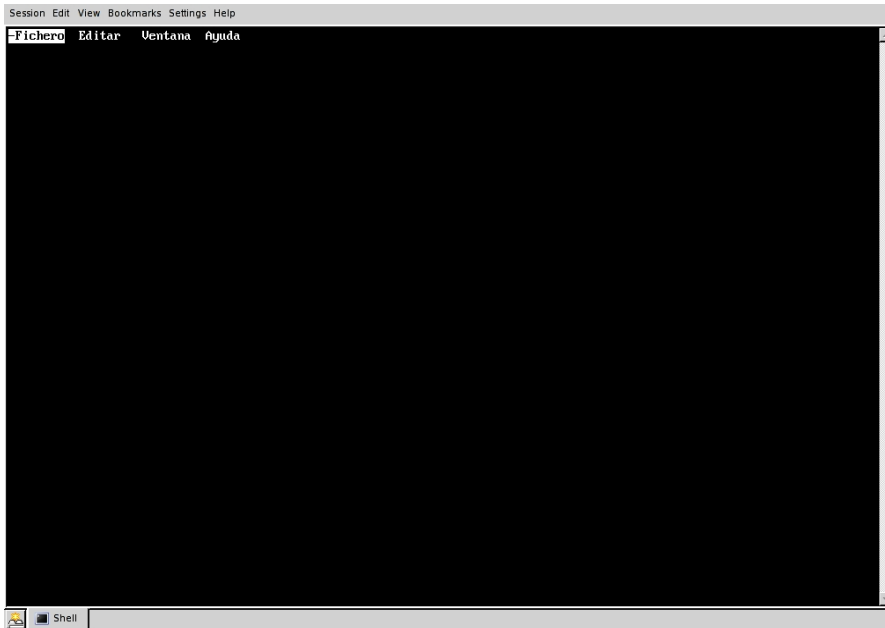


Figura 2: Eso está mejor: una barra de menús, y además...

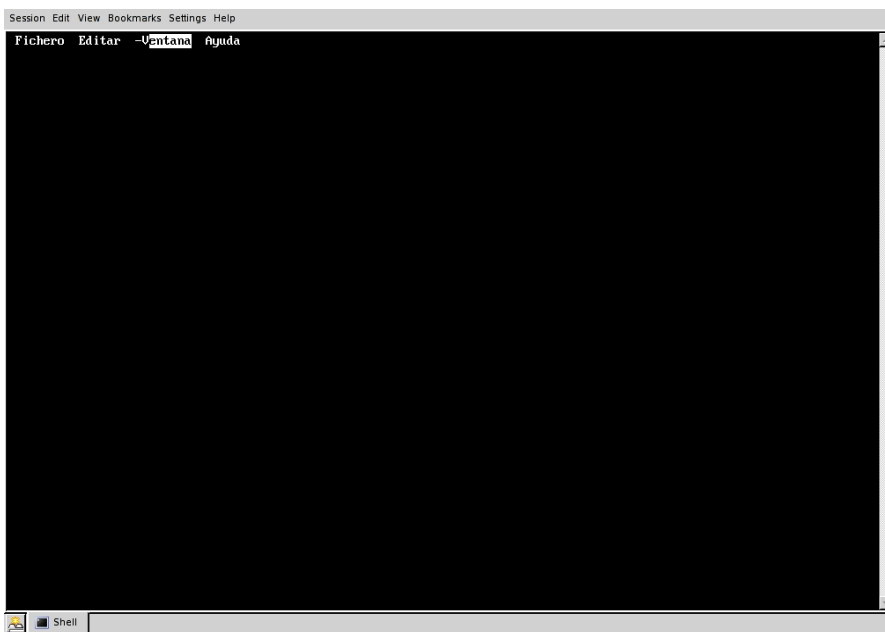


Figura 3: ¡Podemos desplazarnos por ella!

como elementos del menú, para el formato.

Lo siguiente es conseguir mover el enfoque de elemento a elemento. Esto lo conseguimos dentro del bucle principal, consultando a la variable *ch*. Si *ch* contiene el pseudo-carácter *KEY\_RIGHT* (correspondiente a la tecla de flecha de cursor derecha), llamaremos a la función *menu\_driver()* de la librería *menu*. La función *menu\_driver()* es el motor tras la interacción del usuario con el menú una vez creado este. Acepta como entrada el puntero al menú al cual afectarán los

cambios y un carácter o pseudo-carácter que dicta el comportamiento del enfoque del menú.

En el ejemplo que nos ocupa, al pulsar el usuario en las flechas izquierda y derecha, el enfoque se desplaza hacia el elemento de la izquierda o derecha, respectivamente. La tercera opción, permite que *menu\_driver()* busque una coincidencia en la lista de opciones en el caso de que *ch* sea un carácter imprimible.

Por ejemplo, si tenemos el enfoque en el primer elemento del menú ("Fichero"

### Listado 3: application.cpp (fragmento)

```

01 .
02 .
03 .
04 void application::mainMenu()
05 {
06     window
07         main_Menu(COLS-2,3,1,1,TRUE,""
08         );
09     vector<string> mitems;
10     mitems.push_back("Fichero");
11     mitems.push_back("Editar");
12     mitems.push_back("Ventanas");
13     mitems.push_back("Ayuda");
14     menuClass
15         main_App_Menu(mitems,main_Menu
16         .getWHandle(),1,10);
17     app_Window.wWrite(2,10,main_Ap
18     p_Menu.hideMenu());
19 .
20 .
21 .

```

- ver figura 2) y pulsamos la letra "v", *menu\_driver()* desplazará el enfoque a la opción "Ventana" (ver figura 3) y empezará a examinar la segunda letra de todas las opciones que empiecen por "v" a la espera de otro carácter imprimible que le puede permitir depurar la selección. Aquí, en este caso, es una funcionalidad un poco tonta, pero podéis imaginar su utilidad si tenemos una larga lista de nombres sacada, por ejemplo, de una base de datos y podemos ir tecleando las letras del apellido buscado y *menu\_driver()* nos iría acercando al registro buscado.

Ha llegado el momento de crear la clase menú.

### Un Menú con Clase

Ahora es cuando empieza lo interesante y quitamos de en medio todas esas feas funciones. El listado 3 muestra un fragmento de la implementación de la clase *application* (ver los dos capítulos anteriores de esta serie, aparecidas en Linux Magazine 1 y Linux Magazine 2 - el

## Listado 4: Implementación de la clase menuClass

```

01 #include <menuclass.h>
02
03 menuClass::menuClass(vector<string> items, WINDOW*
window=stdscr, int rows=16, int
columns=1)
04 {
05     item_List=items;
06     createItemList();
07     menu_Window=window;
08
09     menu_Thing=new_menu((ITEM**)me
nu_Items);
10
11     set_menu_format(menu_Thing, row
s, columns);
12
13     set_menu_win(menu_Thing, menu_W
indow);
14
15     set_menu_sub(menu_Thing, derwin
(menu_Window, 1, COLS-4, 1, 2));
16
17     showMenu();
18     moveFocus();
19 }
20
21 menuClass::~menuClass()
22 {
23 }
24
25 string menuClass::hideMenu()
26 {
27     string
option=getSelectedItem();
28
29     unpost_menu(menu_Thing);
30     refresh();
31     free_menu(menu_Thing);
32     while (*menu_Items)
33         free_item(*menu_Items++);
34     return option;
35 }
36
37 void
menuClass::createItemList()
38 {
39     unsigned int i;
40     menu_Items=(ITEM**)calloc(item
_List.size(), sizeof(ITEM*));
41
42     for
(i=0; i<item_List.size(); i++)
43         menu_Items[i]=new_item(item_Li
st.at(i).c_str(), "");
44
45     menu_Items[i]=new_item(NULL, ""
);
46 }
47
48 void
menuClass::setMenuWindow(WINDOW
W* window=stdscr)
49 {
50     menu_Window=window;
51     setMenuWindow(menu_Window);
52 }
53
54 void menuClass::showMenu()
55 {
56     post_menu(menu_Thing);
57     wrefresh(menu_Window);
58 }
59
60 void menuClass::moveFocus()
61 {
62     int ch;
63     while((ch=getch())!=13)
64     {
65         switch (ch)
66         {
67             case KEY_RIGHT:
68                 menu_driver(menu_Thing, REQ_RIG
HT_ITEM);
69                 break;
70             case KEY_LEFT:
71                 menu_driver(menu_Thing, REQ_LEF
T_ITEM);
72                 break;
73             case KEY_UP:
74                 menu_driver(menu_Thing, REQ_UP_
ITEM);
75                 break;
76             case KEY_DOWN:
77                 menu_driver(menu_Thing, REQ_DOW
N_ITEM);
78                 break;
79             default:
80                 menu_driver(menu_Thing, ch);
81                 break;
82         }
83         wrefresh(menu_Window);
84     }
85 }

```

código completo se puede descargar de [1]) que muestra el método que activa el menú principal de la aplicación. No requiere mucha explicación. Creamos una ventana donde alojar el menú (esto es práctica habitual en curses) y a continuación utilizamos un vector de cadenas para almacenar las etiquetas de cada opción. Después creamos un objeto *menuClass* al cual pasamos el vector, el handle de la ventana y la posición del menú dentro de esa ventana. He optado por la fila 1 y columna 10 para no “pisar” el borde del cual he dotado la ventana

con el fin de que quede el área del menú claramente delimitado (ver figura 4).

La inicialización de la clase implica la creación y visualización del menú, a la vez que se traslada el enfoque a él. El enfoque permanece con el objeto hasta que el usuario pulsa [Enter] habiendo realizado su selección (todo esto lo vemos un poco más abajo).

Una vez que el menú libera el enfoque, recogemos la salida del método *hideMenu()*, que es la opción escogida cuando el usuario pulsó [Enter] y lo imprimimos en pantalla para demostrar

que es correcto. Obviamente, en un caso real, la clase *application* tendría que hacer algo más productivo con esta salida, como guardar el fichero en edición, abrir una ventana de ayuda, etc.

Por fin, cerramos la ventana del menú para que desaparezca del entorno.

Cómo habréis imaginado, la clase *menuClass* es el meollo de la cuestión. El listado 4 muestra su implementación. Esta clase es virtualmente hermética, es decir casi ninguno de sus métodos ni ninguno de sus atributos son accesibles desde fuera de la clase. Ella se la guisa y

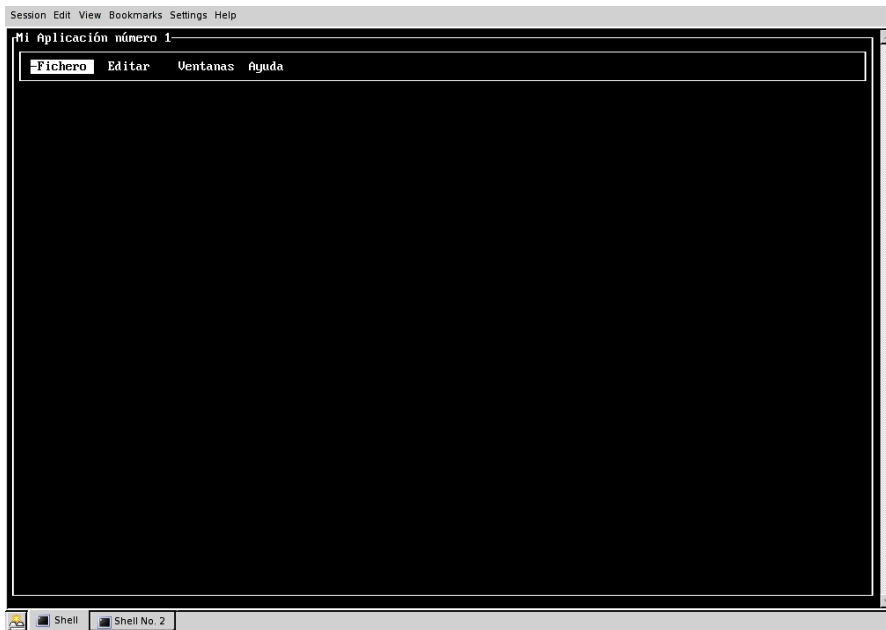


Figura 4: Un menú delimitado en nuestra aplicación.

ella se la come. La excepción son el constructor (a partir de la cual se llaman todos los otros métodos), el destructor (que está vacío), el método *hideMenu()* (que se llama para cerrar el menú y recoger el valor seleccionado) y *getSelectedItem()*, que devuelve la etiqueta de la opción actual escogida (ya que es útil poder acceder a esta información en cualquier momento).

El constructor *menuClass()* (de la línea 3 a la 14) es la encargada de llamar a todos los otros métodos (a excepción de *hideMenu()*) en su debido orden. Primero inicializa un vector *string* llamado *item\_List* que, como su nombre indica, contendrá la lista de elementos que mostrará el menú. Seguidamente, llamamos al método *createItemList()* que hace exactamente eso: crear la lista de elementos en un formato que podrá insertarse en el menú. Si se ha seguido este artículo hasta el momento, el método *createItemList()* (de la línea 34 a la 42) no entrañará ningún misterio: Primero reservamos memoria para la lista (línea 37) y a continuación volcamos las etiquetas en el espacio reservado (líneas 39 y 40), para, finalmente, marcar el final de la lista con un elemento nulo.

De vuelta al constructor, el siguiente paso consiste en inicializar el menú propiamente dicho (línea 8), establecer su formato, la ventana y la subventana donde se va a mostrar y exhibirlo. El encargado de esto último es el método

*showMenu()* (ver de la línea 51 a la 55) que simplemente utiliza la función *post\_menu()* de la librería *menu* y refresca la ventana con *wrefresh()*.

El constructor, por último, llama al método *moveFocus()* que se puede ver en el listado de la línea 57 a la 82. Este método se limita a escuchar las entradas del teclado y mueve el enfoque de un elemento a otro según convenga. Para que se puedan apreciar los cambios, después de que el enfoque haya cambiado, hay que refrescar la ventana (ver línea 80). Si el usuario pulsa [Enter] (clave ASCII 13), el bucle termina.

Por último, *hideMenu()* recoge la última opción escogida y lo coloca en la variable de cadena *option*, esconde el menú y libera el espacio reservado por la lista de elementos. La variable *option* se devuelve a la función que llamó al método para ser procesada.

Todos Juntos En el listado 5 podemos ver una función *main* que hace uso de la clase explicada a través de la clase *application*. Como se puede comprobar, no es nada complejo explotar esta clase. Para mostrar el menú, pulsaremos en F2 y para mostrar una ventana de diálogo como el que vimos el mes pasado, pulsamos en F1.

## Conclusión

En este artículo no hemos hecho más que rayar la superficie de las funcionalidades

## Listado 5: El fichero principal.cpp

```

01 #include "application.h"
02
03 int main()
04 {
05     int ch, cDialogos=1;
06     application my_App(TRUE,"Mi
    Aplicación número %i",1);
07
08     while((ch=my_App.wGetch())!=KEY_
    Y_HOME)
09     {
10         switch (ch)
11         {
12             case KEY_F(1):
13
14                 my_App.dialWin(50,10,"Esto se
                    supone un Diálogo.", "Diálogo
                    %i",cDialogos++);
15                 break;
16             case KEY_F(2):
17                 my_App.mainMenu();
18                 break;
19         }
20     }
21     my_App.~application();
22     exit(0);
23 }

```

de la librería *menu* de *curses*. Existen muchas otras cosas que se pueden hacer. Entre ellas, la posibilidad de asociar una función con cada elemento de menú que se ejecute cuando se selecciona una opción.

En todo caso, si tanto escribir te da pereza, existe por ahí toda una serie de clases (como, por ejemplo, las que se suelen instalar en */usr/share/doc/libncurses5-devel-5.3/c++* con el paquete estándar de *curses*) con sus ejemplos que implementan más funcionalidades, tanto de menús, como formularios y otros aspectos de *curses*. Pero, estate advertido, están mal documentadas y son bastante más complejas que los mías. ■

## RECURSOS

[1] Listados completos en formato tar.gz con Makefile para facilitar la compilación <http://www.linuxmagazine.com.es/Magazine/Downloads/03>