

Interfaces de Texto con *curses*

Dialogando se Entiende la Gente

El otro día estuve en un taller mecánico de esos que tanto me gusta poner como ejemplo (sí, los frenos de mi coche ya funcionan de nuevo, gracias por preguntar) y tuve ocasión de comprobar por enésima vez lo que ya mencioné en esta misma sección hace un mes: ahí los tenías, dos ordenadores Pentium de trigésima generación, corriendo Ventanufilas eXPlosión, chupando miles de megas de RAM, requiriendo tarjetas gráficas de última generación y discos duros del tamaño del desierto de los Monegros y todo ¿para qué? Para que el personal de administración corriese un programa de facturación en modo texto. **POR PAUL C. BROWN**

i Que desperdicio! O tal vez no. Creo que cada vez hay más gente aparte de mí que piensan que para lo único que sirve un Windons es para jugar, ver DVDs y correr programas en modo texto. Es la única manera de estar seguros de que no se cuelgue o se comprometa el sistema con algún fallo de seguridad. A mí entender, comprar un superordenador, más potente que los que se



utilizaron para poner a los primeros hombres en la Luna, y pagarle una licencia a Microsoft para eso, es una burrada. Ya puestos, sirve una instalación de Linux (o BSD) en modo texto, sobre cualquier trasto viejo rescatado del desván y un buen programa en *curses*, que con un poco de trabajo, se puede hacer de todo. Precisamente, el ejercicio de este mes consiste en demostrar lo sencillo que es crear botones en una aplicación de consola. Envolveremos todo en clases C++ y nos basaremos en la aplicación que ya iniciamos el mes pasado. El ejemplo que implementaremos es una ventana de

diálogo con tres botones: “Aceptar”, “Cancelar”, “Abortar”. El usuario ha de ser capaz de navegar por los botones (utilizando, por ejemplo, las teclas *cursor derecha* y *cursor izquierda*) y seleccionar uno de los botones con, por ejemplo, *Enter*. Al pulsar esta última tecla, el diálogo se ha de cerrar y la aplicación ha de poder recuperar el valor del botón pulsado para así actuar en consecuencia. Si os descargáis el fichero *curses.tar.gz* que encontraréis en <http://www.linuxmagazine.com.es/Magazine/Downloads/02> y lo compiláis (se incluye un *Makefile* para simplificar este paso), descubriréis que el programa de ejemplo

funciona como se ha descrito en el párrafo anterior. Una vez con la aplicación corriendo, si pulsamos en *F1* se abre una ventana de diálogo con los tres botones descritos. Podemos movernos de izquierda a derecha o de derecha a izquierda con las teclas de cursor, y se irán iluminando cada uno de los botones (ver Figura 1). Cuando pulsamos *Enter* el diálogo se cierra y la aplicación recoge el valor e imprime el nombre del botón pulsado (ver Figura 2). Para salir de la aplicación y volver a la línea de comandos, pulsaremos *Esc*.

Aplicación

Lo primero es echarle un vistazo a la función *main()*. Como podemos observar en el Listado 1, sigue siendo minimal. Aunque hacemos una llamada a un nuevo método (*dialwin()*) en el bucle principal de la función, que es precisamente la que se encargará de poner un cuadro de diálogo en la pantalla. Como en la vasta mayoría de los casos, el número de botones que contiene un

diálogo son dos, *Aceptar* y *Cancelar* (aunque nuestro ejemplo contenga tres), en aras de la simplicidad y para mantener el código simple, no implementamos ningún método en la clase *application* (ver Listado 2) que sirva para personalizar los botones (esto veremos como se hace en futuras entregas de esta serie). Y ya que hablamos de *application*, veamos como ha cambiado desde el mes pasado. Como podemos ver en el Listado 2, es igualmente muy sencillo y minimal. Primero tenemos un nuevo tipo de dato, el tipo *button*, del que después hablaremos, y con él creamos un vector que almacena los datos de los tres botones, *Aceptar*, *Cancelar* y *Abortar*, que vamos a mostrar. Para aquellos no familiarizados con la clase *vector*, os diré que es una adición muy útil al arsenal de clases de C++. Para utilizarlo, hay que incluir el fichero de cabecera *vector* (con *#include <vector>*). Básicamente, un vector en C++ es un array que puede admitir cualquier tipo de dato, primitivo o no, e implementa una serie de útiles métodos para añadir nuevos elementos, devolver elementos de una posición determinada (con el método *at(índice)*, del que después veremos un ejemplo) o para calcular su tamaño tanto en bytes, como en número de elementos. Para llenar el vector utilizamos el método *push_back()* asociado a la clase *vector* que lo que hace es añadir un elemento a la cola del vector. El siguiente paso consiste en crear un objeto *window* (en este caso lo llamamos *dial_box*) y, como tenemos sobrecargado el constructor de la clase *window* (después lo veremos), al pasarle como parámetro el vector de botones, entre otros, se invoca al constructor que crea una ventana de diálogo. Básicamente eso es todo, puesto que, una vez llamada, el control se pasa a la ventana de diálogo, que implementa su propio bucle y que no cede el control de vuelta a *application* hasta que se seleccione uno de los botones y se pulse *Enter*. Cuando esto sucede, *application* puede consultar el botón activo en el momento de cerrar la ventana de diálogo recogiendo el valor devuelto por el método *closeWindow()*. El valor devuelto por *closeWindow()* es un número, que en este caso puede ser 0 (*Aceptar*), o 1 (*Cancelar*), o 2 (*Abortar*). En nuestro

ejemplo, el método *dialWin()* se limita a imprimir la etiqueta del botón para efectos de depurado.

Ventana

Pasemos ahora a los métodos de la clase *window*, que son los que hacen la mayor parte del trabajo sucio. Podemos ver un fragmento del archivo *window.cpp* en el Listado 3. Lo primero que destaca es el constructor que implementa las ventanas de diálogo. Tenemos un atributo,

Listado 1: El método dialwin() en application.cpp

Listado 1: principal.cpp

```
01 #include "application.h"
02
03 int main()
04 {
05     int ch, cDialogos=1;
06     int counter=0;
07     application my_App(TRUE,"Mi
    Aplicación número %i",1);
08
09     while((ch=my_App.wGetch())!=27
10 )
11     {
12         my_App.wWrite(1,2,"Vuelta
    n°: %i",counter++);
13         if(ch==KEY_F(1))
14         {
15             my_App.dialWin(50,10,"Esto se
    supone un Diálogo.", "Diálogo
    %i",cDialogos++);
16         }
17     }
18
19     my_App.~application();
20     exit(0);
21 }
```

```
01 void application::dialWin(int
    dial_Width, int dial_Height,
    string dial_Text, string
    dial_Title="Untitled",...)
02 {
03     button
        the_Button(1,dial_Height-4,"Ac
        eptar");
04     vector<button> dial_Buttons;
05
06     dial_Buttons.push_back(the_But
        ton);
07
08     the_Button=button(11,dial_Heig
        ht-4,"Cancelar");
09
10     dial_Buttons.push_back(the_But
        ton);
11
12     the_Button=button(22,dial_Heig
        ht-4,"Abortar");
13
14     dial_Buttons.push_back(the_But
        ton);
15
16     makeString(dial_Title);
17     window dial_Box(dial_Width,
        dial_Height, dial_Text,
        dial_Buttons, 0, dial_Title);
18
19     int
        button_Pressed=dial_Box.closeW
        indow();
20
21     app_Window.wWrite(2,2,dial_But
        tons.at(button_Pressed).getTag
        ());
22 }
```

Listado 2: Los métodos relevantes de window en window.cpp

```

001 #include "window.h"
002
003 .
004 .
005 .
006
007 window::window(int wide,int high,string
    content,vector<button> buttons,int button_Num=0,
    string win_Name="Untitled"... )
008 {
009     w_Buttons=buttons;
010     borders_Inactive.push_back(ACS_ULCORNER);
011     borders_Inactive.push_back(ACS_URCORNER);
012     borders_Inactive.push_back(ACS_LLCORNER);
013     borders_Inactive.push_back(ACS_LRCORNER);
014     borders_Inactive.push_back(ACS_HLINE);
015     borders_Inactive.push_back(ACS_VLINE);
016
017     borders_Active.push_back(ACS_ULCORNER|A_STANDOUT
    );
018     borders_Active.push_back(ACS_URCORNER|A_STANDOUT
    );
019     borders_Active.push_back(ACS_LLCORNER|A_STANDOUT
    );
020     borders_Active.push_back(ACS_LRCORNER|A_STANDOUT
    );
021     borders_Active.push_back(ACS_HLINE|A_STANDOUT);
022     borders_Active.push_back(ACS_VLINE|A_STANDOUT);
023
024     active_Button=button_Num;
025     if(wide>COLS || high>LINES)
026     {
027         makeDialogue(20,10,"Dialogue too
    big","Error");
028         closeWindow();
029     }
030     makeString(win_Name);
031     makeDialogue(wide,high,content,win_Name);
032 }
033
034
035 int window::closeWindow()
036 {
037     del_panel(p_Handle);
038     delwin(w_Handle);
039     refresh();
040     return (active_Button);
041 }
042
043 .
044 .
045 .
046
047 void window::makeDialogue(int wide,int
    high,string content,string win_Name)
048 {
049     int posX=(COLS/2)-(wide/2);
050     int posY=(LINES/2)-(high/2);
051
052     w_Handle=newwin(high,wide,posY,posX);
053
054     title=win_Name;
055     has_Frame=TRUE;
056
057     p_Handle=new_panel(w_Handle);
058
059     putTitle();
060     wWrite(0,0,content);
061
062     activeButton();
063
064     showPanels();
065     dialLoop();
066 }
067
068 void window::dialLoop()
069 {
070     int ch;
071
072     while((ch=getch())!=13)
073     {
074         switch (ch)
075         {
076             case KEY_LEFT:
077                 if(active_Button==0)
078                     active_Button=w_Buttons.size();
079                 active_Button--;
080                 activeButton();
081                 break;
082             case KEY_RIGHT:
083                 active_Button++;
084                 if(active_Button==w_Buttons.size())
085                     active_Button=0;
086                 activeButton();
087                 break;
088             }
089         //wWrite(0,1,"Botón %i",active_Button);
090         //wWrite(0,2,"Pulsado %i",ch);
091     }
092 .
093 .
094 .
095
096 void window::activeButton()

```

Listado 2: Los métodos relevantes de window en window.cpp (continuación)

```

097 {
098     for(unsigned int i=0;i<w_Buttons.size();i++)
099     {
100         drawButton(w_Buttons.at(i).getTag(),w_Buttons.at
101                   (i).getPx(),
102                   w_Buttons.at(i).getPy(),borders_Inactive);
103     }
104     drawButton(w_Buttons.at(active_Button).getTag(),
105               w_Buttons.at(active_Button).getPx(),
106               w_Buttons.at(active_Button).getPy(),
107               borders_Active);
108 }
109 void window::drawButton(string label,int but_X,
110                          int but_Y, vector<chtype> borders)
111 {
112     //Primera fila del botón
113     mwaddch(w_Handle, but_Y, but_X, borders.at(0));
114     for(unsigned int i=0;i<label.length();i++)
115     {
116         mwaddch(w_Handle, but_Y+1, but_X+1+i, borders.at(4));
117         mwaddch(w_Handle, but_Y+1, but_X+label.length()+1,
118                 borders.at(5));
119     }
120     //Tercera fila del botón
121     mwaddch(w_Handle, but_Y+2, but_X, borders.at(2));
122     for(unsigned int i=0;i<label.length();i++)
123     {
124         mwaddch(w_Handle, but_Y+2, but_X+1+i, borders.at(4));
125     }
126     mwaddch(w_Handle, but_Y+2, but_X+label.length()+1,
127             borders.at(3));
128     wrefresh(w_Handle);
129 }
130 .
131 .
132 .

```

w_Buttons, que contiene el vector de botones de la ventana de diálogo. A continuación, los atributos *borders_Inactive* y *borders_Active* guardan el estilo de los bordes del botón en modo inactivo (botón no seleccionado) y modo activo (botón seleccionado), respectivamente. Yo he escogido mostrar los bordes normales en el primer caso y en vídeo inverso en el segundo. El efecto se puede apreciar en la Figura 1 donde se ve el botón central seleccionado. La librería *curses* implementa una serie de *seudo-caracteres* para dibujar líneas y bordes, y estos son los que utilizo aquí para crear el canto de los botones. Por ejemplo, *ACS_ULCORNER* es un carácter que se utiliza en la esquina superior izquierda (UL = Upper, Left) de un cuadro y *ACS_HLINE* es una línea horizontal. El atributo *active_Button* contiene el valor que le dice a la clase

qué botón está activo. Seguidamente, el constructor comprueba el tamaño de la ventana de diálogo, y si es demasiado

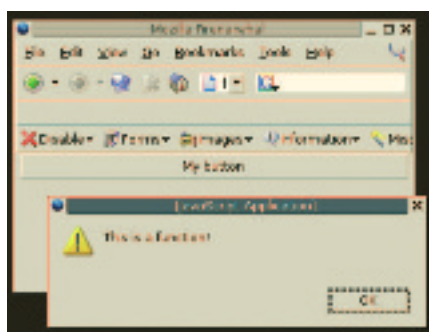


Figura 1: Se puede navegar por los botones utilizando las teclas de cursor.

grande, imprime un error. Este diálogo de error mostrará los botones que hemos descrito para la ventana de diálogo que pretendíamos crear, por tanto, puede ser demasiado pequeño para mostrarlos

todos. Otra cosa es que, tal y como viene implementado, su comportamiento no es del todo funcional, exigiendo que el usuario pulse dos veces *Enter* para volver a la ventana principal de la aplicación. Pero, como el motivo de su presencia es meramente a efectos de depurado (nunca nos va a interesar que un usuario vea esta ventana de error), así se queda, como doble recordatorio de lo malos que son los bugs en versiones de producción de un programa. Después de este constructor un tanto especial, tenemos el método *closeWindow()*, que el mes pasado se limitaba a cerrar la ventana y este mes, además, devuelve un valor: el número del botón pulsado. El método *makeDialogue()* es el que se encarga de colocar la ventana del diálogo en la pantalla. Lo primero que hace es calcular un posición más o

Listado 4: La clase `button` en `button.cpp`

```

01 #include "button.h"
02
03 button::button(int pos_X=0,int
    pos_Y=0,string
    label="Aceptar")
04 {
05     px=pos_X; py=pos_Y;
06     tag=label;
07
    button_Length=tag.length()+2;
08 }
09
10 button::~button()
11 {
12
13 }

```

menos central para nuestro diálogo. A continuación crea la ventana `curses` que contendrá la ventana de diálogo, le coloca un título, un borde, y lo pone sobre un panel para poder apilarlo encima de otras ventanas. Después se invoca a `activeButton()`, que es el método encargado de dibujar (por medio del método `drawButton()`) cada uno de los botones, primero sin enfoque, y después redibuja el botón que tiene el enfoque con los bordes resaltados. A continuación, `makeDialogue()` muestra la ventana, actualizando la pantalla y llama a `dialLoop()`, el método que contiene el bucle del diálogo. Prosigamos con `dialLoop`, que es el siguiente en orden de aparición. Este método es un sencillo bucle que comprueba en cada pasada si se ha pulsado la tecla `Enter` para salir. Si no es así, mira si las teclas pulsadas son las de cursor izquierda (si es así, suma 1 al atributo `active_Button`) o cursor derecha (en tal caso resta 1 al atributo `active_Button`). A continuación, redibuja los botones, iluminando el botón activo. El redibujado de los botones lo realiza la función `activeButton()` que itera por el vector apagando todos los botones (dibujando el borde inactivo). El bucle utilizado es otro buen ejemplo de la flexibilidad de los vectores: la variable `i` inicialmente vale cero y va incrementándose hasta el límite del tamaño del vector, que podemos averiguar con el método de la clase `vec-`

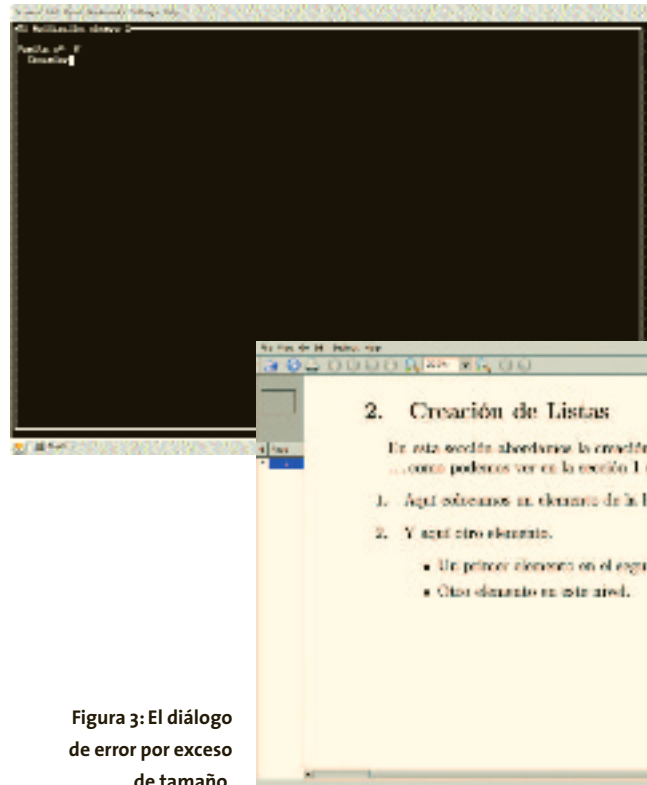


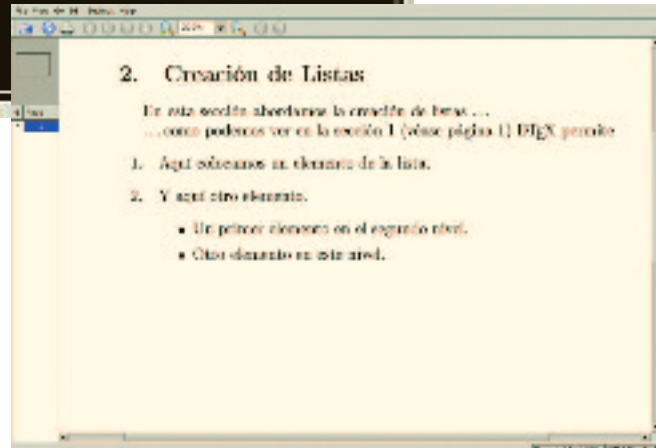
Figura 3: El diálogo de error por exceso de tamaño.

`tor, size()`. Después, utilizamos `i` para poder referirnos a cada uno de los elementos del vector de botones con el método `at()`, explicado más arriba. A continuación, volvemos a utilizar `at()` para dibujar un borde activo alrededor del botón activo, utilizando como índice `active_Button`. Y llegamos al método del que todos estaban hablando: `drawButton()`. Este método parece complejo, pero en realidad es una tontería. Un botón normalmente constará de tres filas de caracteres ¿vale? El borde superior, la parte central con el texto de la etiqueta y el borde inferior. `drawButton` se limita a dibujar cada una de esas filas con el borde con el atributo (activo o inactivo) correspondiente.

Botón

Lo difícil se ha acabado, pero nos queda una pequeña clase antes de poder dar carpetazo al artículo. En el Listado 4 podéis ver la sencillísima clase `button` que almacena en su interior la información necesaria (la posición del botón y la etiqueta a mostrar) para que podamos dibujar cada uno de los botones. ¿Que de tan sencilla que es esta clase podría haber sido una estructura? Pues sí, pero preferí hacerlo clase, no fuera que más adelante se me ocurriera funcionalidades

Figura 2: La aplicación muestra el nombre del botón pulsado.



extra. Por ejemplo, como alguno ya sabrá, es posible pasar una función C como parámetro de otra función. ¿No sería éste un mecanismo ideal para asociar una acción con un botón? Tal vez en otro capítulo.

Conclusión

Hemos visto lo relativamente sencillo que es implementar botones en `curses` y como podemos hacer que afecten al resto del entorno. El siguiente paso iría dirigido a coger el valor devuelto por la ventana de diálogo, a través de una estructura `switch... case`, llevar a cabo las distintas acciones para cada caso. El mes que viene veremos como crear menús para nuestra aplicación y con ello haremos abordado ya la mayor parte de la implementación de interfaces con `curses`. Hasta entonces, disfrutad de la programación. ■

EL AUTOR

Paul C. Brown lleva ya unos cuantos años escribiendo sobre tecnología e informática, especializándose en GNU/Linux y tecnologías de Software Libre. En la actualidad comparte la dirección de *Linux Magazine España* y la escritura con la programación recreativa y la ardua tarea de enseñar *Squeak* a su hijo de cinco años.