UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Estrategias alternativas para mostrar información visual usando el texto y el cursor

Apellidos, nombre	**Agustí i Melchor, Manuel,** (magusti@disca.upv.es
Departamento	**Departamento de Informática de Sistemas y Computadores**
Centro	**Escola Tècnica Superior d'Enginyeria Informàtica** Universitat Politècnica de València

1 Resumen

El desarrollador de aplicaciones se encuentra, en ocasiones, en una situación poco habitual: ha de diseñar y realizar una aplicación que no recibe el foco de atención del usuario. Entonces, ¿de qué manera puedes mostrar información visual y que no quede tapada por otras ventanas? Es el caso de aplicaciones que se encargan de monitorizar lo que acontece en nuestros computadores y que suelen estar en forma de tareas en segundo plano, a la espera de que se produzca un evento que las active o directamente a petición del usuario. Son las que se conocen como notificaciones del sistema.

Así que en este documento nos vamos a centrar en ver **ejemplos de uso de librerías para mostrar información sin abrir una nueva ventana**. La Figura 1 muestra ejemplos de lo que estamos proponiendo:

a) El cursor puede cambiar de forma y/o color para informar al usuario.

b) Pequeñas ventanas, a modo de "bocadillos" de las viñetas gráficas que aparecen habitualmente en la barra del sistema y desaparecen al poco.

c) Mensajes superpuestos a todas las ventana, cuyo fondo transparente permite seguir viendo lo que estamos haciendo..

Figura 1: Ejemplos de uso de (a) ventanas con formas de cursores, (b) notificaciones y (c) mensajes superpuestos en pantalla.

La exposición se centrará en la plataforma Linux/Unix, pero existen las mismas herramientas o equivalentes en otros sistemas operativos. Veremos el uso de librerías para ser utilizadas dentro de aplicaciones realizadas en código *C* y que se apoyan directamente sobre el gestor de ventanas *X Window* con la librería *Xlib* [1]. De esta manera lo que se va a mostrar es directamente portable a diferentes distribuciones de Linux.

2 Objetivos

Una vez que el lector haya explorado los ejemplos que aquí se muestran, será capaz de:

- Mostrar texto e indicaciones gráficas sobre todas las ventanas.
- Cambiar la apariencia del cursor en pantalla.
- Mostrar mensajes emergentes en la barra de tareas del sistema.

Aunque se puede leer este documento y ver si interesa el uso de estas técnicas para sus aplicaciones, la diversión está en probarlas y modificarlas, Será la mejor manera de comprobar si sirven. Anímate y ve preparando un terminal y un editor de código. Por brevedad en la exposición, estará centrada en algunas funciones, no se va a hacer un recorrido exhaustivo de todas las

posibilidades del interfaz (también llamado API[1]) de *XWindow*. Eso sí, te dejaré ejemplos listos para seguir explorando tú mismo. Cuento con que tengas ganas de probar, no solo de leer. ¿Empezamos?

3 Introducción

En Unix, la interfaz gráfica de usuario no es parte del sistema operativo y ha venido siendo construido sobre el protocolo *X (*también conocido como *X Window* o *X.org*), implementado a través de un servidor y utilizando las primitivas que proporciona *Xlib*. Hay que señalar que, como muestra la Figura 2 [1], existen otras opciones, puesto que algunas versiones de Linux/Unix están utilizando XCB como alternativas a *Xlib*.

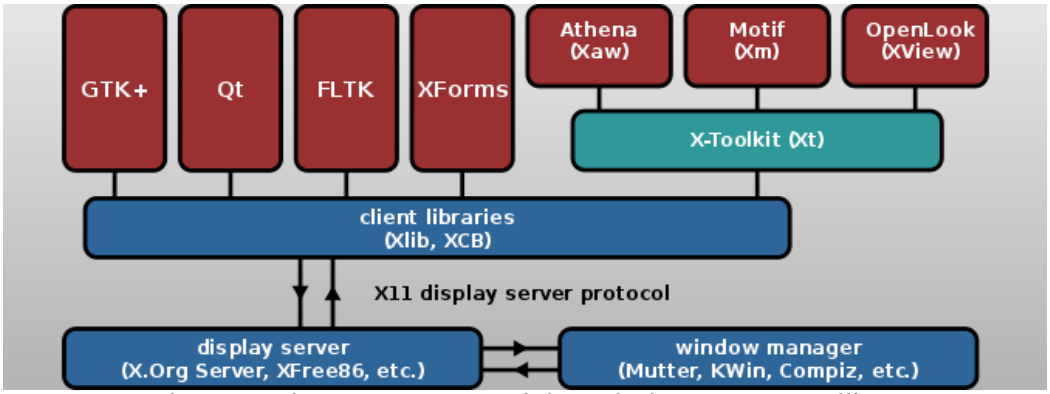


Figura 2: Arquitectura de componentes del X Window System: Xlib y XCB. Imagen de [1].

Además, algunos gestores de ventanas como GNOME o KDE utilizan las librerías *GTK+* o *Qt*[2], respectivamente, y ofrecen acciones similares a las que veremos aquí. Si no queremos sobrecargar nuestra aplicación con el peso de uno de estos dos potentes SDK[3], entre otras cosas porque, recuerda, estamos proponiendo alternativas para aplicaciones con un interfaz mínimalista; entonces la solución ha de usar directamente primitivas de *X Window* [1]. Esto nos lleva a utilizar Xlib o XCB, dada la alta posibilidad de encontrarnos con la primera, vamos a quedarnos con ella como elección inicial.

También queda fuera de este texto abordar soluciones propias de un entorno como *Plasma* en KDE[4], que pueden ofrecer opciones que no poseen las librerías que se exploran aquí y que solo se podrían utilizar en ese gestor. Tampoco se van a explorar otras especificaciones como *Wayland* y *Mir*[5] que están proponiendo a OpenGL[6] como base en lugar de *Xlib* o *XCB*. Es pronto todavía para decir cual se quedará.

---

1 API <https://en.wikipedia.org/wiki/Application_programming_interface>.

2 Más detalles sobre ellas en <http://www.gtk.org/> y <http://www.qt.io/>, respectivamente.

3 SDK <https://en.wikipedia.org/wiki/Software_development_kit>.

4 Veasé Plasma/Notification en *KDE Community Wiki* <https://community.kde.org/Plasma/Notifications>.

5 Ambas se puede ver en sus respectivas páginas: <http://wayland.freedesktop.org/> y <https://wiki.ubuntu.com/Mir>.

6 Se puede consultar al respecto de esto en “How is the linux graphics stack organised?” en < https://unix.stackexchange.com/questions/7943/how-is-the-linux-graphics-stack-organised/178420#178420>.

4 Desarrollo

Veamos ahora cada librería por separado con un ejemplo, destacando las operaciones más relevantes para utilizarlas, así como las instrucciones para su instalación y compilación.

4.1 *On Screen Display* (OSD)

La idea de esta técnica es poner información de texto sobre las ventanas de todas las aplicaciones en marcha, al estilo de como suele verse en los televisores, esto es, con el fondo del área de la visualización transparente, para no tapar demasiado y captar la atención del usuario.

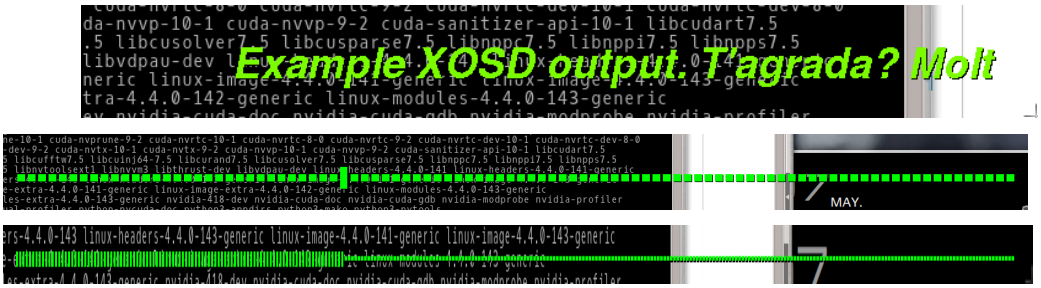


Figura 3: Ejemplos de uso de mensajes superpuestos en pantalla.

Se propone para esta estrategia utilizar la librería XOSD[7] que ofrece operaciones de texto y dos modos para representar un control de desplazamiento, como muestra la Figura 3. Para instalarla, bastará ejecutar esta orden

```
$ sudo apt-get install libxosd-dev
```

Y para ilustrar su uso veamos un ejemplo en el Listado 1 (que he llamando *osd_example.c*) y que se puede compilar con una línea como:

```
$ gcc osd_example.c -o osd_example -lxosd
```

El código muestra un ejemplo de cada una de las tres categorías: `XOSD_string`, `XOSD_slider` y `XOSD_percentage`. Observa que los tres casos parten de la creación de una variable de tipo *xosd*, que es inicializada con los mismos valores de atributos de estilo y el tiempo que va a permanecer el mensaje en pantalla. Cuando se quiere mostrar el mensaje hay que utilizar

```
xosd_display (osd, 0, XOSD_string, "Example XOSD output. T'agrada? Molt").
```

Este es el primer caso del Listado 1, que escribe una cadena de texto en pantalla, consiguiendo que se muestre el primer ejemplo de la Figura 3. Los otros dos ejemplos de XOSD son análogos a este en cuanto a las instrucciones de formato, así que las he dejado indicadas para hacer más corto el código mostrado. Solo cambia la invocación de `xosd_display`, puesto que el tercer y cuarto argumentos dependen del tipo de objeto a mostrar. En los dos casos que quedan por ver es un entero, entre 0 y 100, que será la posición relativa que se utilizará para posicionar el *slider* (en el caso de la barra de desplazamiento, `XOSD_slider`, que corresponde a la imagen central de la Figura 3) o a la parte ocupada en el caso de la barra de porcentaje (`XOSD_percentaje`, que es la imagen de abajo de la Figura 3).

7 De forma breve se puede ver en “xosd(3) - Linux man page”. Disponible en línea en <https://linux.die.net/man/3/xosd>.

```c
#include <stdlib.h>
#include <xosd.h>

int main (int argc, char *argv[]) {
    xosd *osd;

    osd = xosd_create (1);
    xosd_set_font(osd, "-adobe-helvetica-*-*-*-*-34-240-*-*-p-*-*-*" );
    xosd_set_colour(osd, "LawnGreen");
    xosd_set_timeout(osd, 3);
    xosd_set_shadow_offset(osd, 1);
    xosd_set_pos( osd, XOSD_middle );
    xosd_set_align( osd, XOSD_center );
    xosd_display (osd, 0, XOSD_string,
                  "Example XOSD output. T'agrada? Molt");
    xosd_wait_until_no_display(osd);
    xosd_destroy(osd);

    osd = xosd_create (1);
    // Copiar aquí las instrucciones de formato del anterior
    xosd_display (osd, 0, XOSD_slider, 33);
    xosd_wait_until_no_display(osd);
    xosd_destroy(osd);

    osd = xosd_create (1);
    // Copiar aquí las instrucciones de formato del primer ejemplo
    xosd_display (osd, 0, XOSD_percentage, 33);
    xosd_wait_until_no_display(osd);
    xosd_destroy(osd);

    return EXIT_SUCCESS;
}
```

Listado 1: Ejemplo de uso de XOSD (osd_exemple.c).

4.2 Cursor

El cursor que muestra la posición del ratón, además, comunica qué acción estamos llevando a cabo con él. Así estamos acostumbrados a verlo en forma de flecha, en forma de barra vertical en un editor o en un terminal, en alguna animación mientras se realiza alguna tarea que lleva un tiempo para indicarnos que debemos esperar (como por ejemplo la carga de un archivo grande). Pero también podemos cambiar la forma que muestra y los colores que utiliza para indicar otros estados o mensajes al usuario.

Existen librerías de funciones que proporcionan operaciones para crear, redimensionar, colorear, modificar la forma o liberar recursos asociados con el uso del puntero en pantalla. Siguiendo con la propuesta de mantener una versión lo más portable a diferentes plataformas, proponemos el uso de *Xlib* [4]. Veamos cómo se utiliza a partir de un ejemplo de código[8] mínimo de aplicación con salida gráfica que utiliza *Xlib*, véase el Listado 2, donde hemos dejado indicado con números entre paréntesis dónde introduciremos las operaciones de manejo del cursor, para la creación de una aplicación con salida gráfica sobre *Xlib.* El ejemplo de este apartado (que se llama *window_creation__xlib.c*) se puede compilar con una línea como:

```
$ gcc window_creation__xlib.c -o window_creation__xlib -lX11
```

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// (1) Nuevos ficheros de cabecera
int main(void) {
   Display *d; Window w;  XEvent e; const char *msg = "Hello, World!";
   // (2) Nuevas variables
   d = XOpenDisplay(NULL);
   if (d == NULL) { fprintf(stderr, "Cannot open display\n");exit(1);}
   s = DefaultScreen(d);
   w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 100, 100, 1,
                           BlackPixel(d, s), WhitePixel(d, s));
   XSelectInput(d, w, ExposureMask | KeyPressMask);
   XMapWindow(d, w);
   //  (3) Nuevas inicializaciones de objetos
   while (1) {
      XNextEvent(d, &e);
      if (e.type == Expose) {
         XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);
         XDrawString(d, w, DefaultGC(d, s), 10, 50, msg, strlen(msg));
      }
      if (e.type == KeyPress)  break; // (4) Nueva gestión de eventos
   }
   XCloseDisplay(d);
   return 0;
}
```

Listado 2: Ejemplo de creación de una ventana con Xlib.

8 Lo tienes en “Window creation / X11 - Roseta Code” disponible en <https://rosettacode.org/wiki/Window_creation/X11#C>.

Si prueba el ejemplo del Listado 2, verá que abre una ventana con un cuadradito en negro y un mensaje “Hello, World!”. Si llevas el cursor hasta esa ventana verás que sigue siendo el puntero con forma de flecha, que es el habitual. Vamos a jugar con él.

Figura 4: Ejemplos de uso de las operaciones de gestión del cursor.

```
// (1) Nuevos ficheros de cabecera
#include <X11/Xutil.h> // XLookupString
#include <X11/cursorfont.h>
// (2) Nuevas variables
   int s, len, formaCursor = 0; char s1[128], buf[128] = {0}; Cursor c;

   XColor foreground_color, background_color;
   KeySym keysym;
//  (3) Nuevas inicializaciones de objetos
   c = XCreateFontCursor(d, formaCursor ); //XC_xterm);
   XDefineCursor(d, w, c);
   foreground_color.red = foreground_color.green = foreground_color.blue = 0 ;
   background_color.red = 65535;
   background_color.green =  background_color.blue = 0 ;

// (4) Nueva gestión de eventos
    if (e.type == KeyPress)    {
      len = XLookupString(&e.xkey, buf, sizeof buf, &keysym, NULL);
      if (keysym == XK_Escape) break;
      else {
       formaCursor +=2;  XFreeCursor(d, c );
       c = XCreateFontCursor(d, formaCursor ); //XC_xterm);
       XRecolorCursor(d, c, &foreground_color, &background_color);
       XDefineCursor(d, w, c);
       sprintf(s1, "Cursor %03d, color", formaCursor);
       DrawString(d, w, DefaultGC(d, s), 10, 70,
                                    "                                ", 25);
       XDrawString(d, w, DefaultGC(d, s), 10, 70, s1, strlen(s1));
      }
   }
```

Listado 3: Modificaciones a introducir en el ejemplo de uso del cursor.

Para ello el Listado 3 agrupa las modificaciones a introducir en este ejemplo básico y que darán pie a que, cada vez que se pulse una tecla en esa ventana, cambie la forma del cursor. Varios de estos posibles cambios se muestra en la Figura 4. ¡Observa que ha cambiado tanto la forma, como el color!

Parte de las nuevas instrucciones son propias del uso del código. Así bajo la etiqueta "(1) Nuevos ficheros de cabecera" se agrupan las nuevas cabeceras que proporcionan la declaración de las nuevas funciones que vamos a emplear. En "(2) Nuevas variables" se recogen las variables necesarias para las instrucciones que se introducen.

Con la etiqueta "(3) Nuevas inicializaciones de objetos" se define un nuevo cursor con la función `XcreateFontCursor`, a partir de la elección del identificador de forma y color deseados. Y, para liberar recursos, utilizaremos después `XfreeCursor`. He dejado el cambio de color del cursor para el bucle principal, así se verá que puedo hacer una cosa o la otra y, también, las dos juntas. Observemos que el caso del color tenemos control para variar las componentes de RGB de fondo y de primer plano con `XRecolorCursor`. Es un buen momento para que experimentemos con valores de nuestra elección.

Finalmente la etiqueta "(4) Nueva gestión de eventos" recoge la variación del bucle principal que hace cambiar el cursor tras cada pulsación de una tecla, si tenemos paciencia, se irá viendo pasar las 77 formas que hay-[4]. Todas toman índices pares, los impares son para definir la máscara de puntos de primer plano y de fondo de la imagen del cursor. Por ese motivo, en el "else" del final del Listado 4, cada vez que ser produce un evento `KeyPress` y no se ha pulsado la tecla *Escape*, se incrementa el índice y se actualiza la forma y color de un nuevo cursor.

4.3 Mensajes emergentes: NotifyOSD

La idea de esta técnica es aprovechar que los usuarios se están acostumbrando a ver aparecer mensajes o notificaciones que aparecen junto a la barra de estado y que, al tiempo, desaparecen. Es por este motivo que también reciben el nombre de burbujas (*bubbles*), *desktop notifications* o mensajes emergentes (*pop-ups*).

Esta estrategia depende del gestor de gráficos que se utilice, en este caso voy a proponer el uso de *NotifyOSD*[9] por estar basado en la especificación de *freedesktop.org*[10] y por su portabilidad a aplicaciones sobre C/C++/C# y Python. Haremos uso desde dentro de una aplicación en lenguaje C, por coherencia con el resto de ejemplos de este documento y porque también se puede utilizar esta estrategia en un *shell-script*[11].

NotifyOSD [5] implementa las instrucciones para declarar un objeto, averiguar las opciones del servidor que atiende al servicio, inicializar las notificaciones y comprobar si están activas e incluso modificar su contenido. Para instalarla, bastará ejecutar esta orden

```
$ sudo apt-get install libnotify-dev
```

Y para ilustrar su uso veamos una ampliación de *[6]* (aparece en el Listado 4 y lo referencio como el fichero *notify-ejemplo1.c*). Para compilarlo se ha de hacer uso de una línea como:

---

9 Disponible en <https://wiki.ubuntu.com/NotifyOSD>.

10 *Desktop Notifications Specification, disponible en* <http://www.galago-project.org/specs/notification/0.9/>.

11 Véase "Notify-send HOWTO" <https://ubuntuforums.org/showthread.php?t=1411620> y "Notify-send through C program " <https://ubuntuforums.org/showthread.php?t=2053678 >.

```
$ gcc  notify-ejemplo1.c -o notify-ejemplo1 `pkg-config --cflags --libs libnotify`
```

El código muestra un ejemplo muy sencillo, que he ampliado para, como muestra la Figura 5, ofrecer un par de ejemplos de uso avanzado que suelo utilizar:

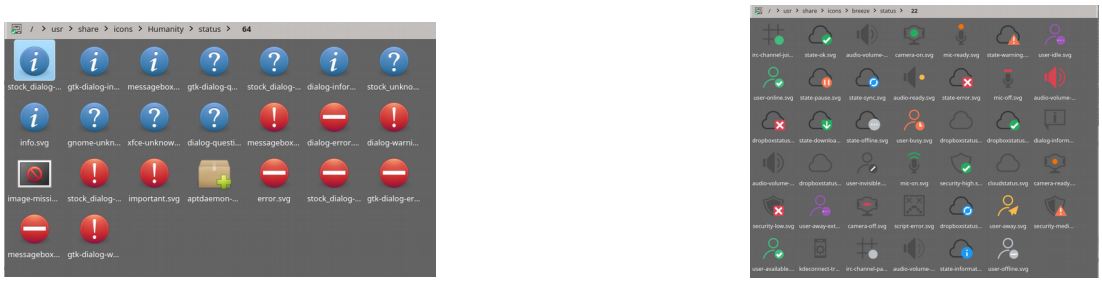
- Por un lado (Figura 5a) está la opción de escoger un icono[12] para hacer rápidamente visible la "categoría" del mensaje (información, aviso, error, ...) o la "aplicación" que lo ha generado.
- Por otro (Figura 5b) el contenido de texto puede estar formateado con negritas, cursiva y, lo más interesante, incluir enlaces para poder hacer clic sobre ellos. Para ello se utilizan las etiquetas habituales de una página web.

```c
#include <libnotify/notify.h>
#include <stdio.h>
int main(int argc, char * argv[] ) {
    notify_init( argv[0]);
    NotifyNotification* n = notify_notification_new ("Hello world",
                                "some message text... bla bla",
                                 0);
    notify_notification_set_timeout(n, 2000); // 2 segons

    if (!notify_notification_show(n, 0))     {
        printf("show has failed\n");        return -1;
    }
    sleep(3); //espera a que se vaya
    // Es posible reutilizar un objeto ya creado
    notify_notification_set_urgency( n,  NOTIFY_URGENCY_CRITICAL);
    notify_notification_update (n,
   "<b>Titol</b>",
   "<i>Enlace URL</i>: <a href=\"https://wiki.ubuntu.com/\"> N.D.G.</a>",
   "/usr/share/icons/breeze/status/22/state-ok.svg" );
    sleep(3); //espera a que s'en vaja
    return 0;
}
```

Listado 4: Ejemplo de uso de mensajes emergentes con libnotify (fichero notify-ejemplo1.c) y que basa en el de [6].

12 Ejemplos de iconos para incluir en los mensajes emergentes. Los tienes disponibles en tu instalación p. ej. en /usr/share/icons/.



a)

b)

Figura 5: Ejemplos de uso avanzados de notificaciones o mensajes en la barra de tareas: catálogo de iconos (a) y ejemplos de texto con formato (b).

5 Conclusión

A lo largo de este documento el lector ha podido ver algunas estrategias para mostrar información sin abrir una ventana. Esto es especialmente útil si no podemos contar con que siempre habrá una ventana de nuestra aplicación en primer plano y que el usuario la está atendiendo continuamente. Al contrario, le dejamos que esté haciendo lo que quiera y, solo cuando sea necesario, le haremos llegar la información:

a) Mostrando un mensaje sobrepuesto a la pila de ventanas.

b) Haciendo cambiar la forma y/o el color del cursor.

c) Mostrando un mensaje que se ubicará siempre sobre la barra de tareas del sistema.

Espero que te hayas animado a probar los ejemplos que he ido mostrando. Depende lo que quieras mostrar y de lo que estés haciendo, una estrategia u otra será la más adecuada. Habrás de experimentar para ver lo sencillo que es y lo profesional que queda hacer uso de ellas en una aplicación.

6 Bibliografía

[1] Gettys, J y Scheifler, R. W. (2002). Xlib − C Language X Interface. X Window System Standard. X Version 11, Release 6.7. The Open Group. X.Org Foundation. Disponible en <https://www.x.org/docs/X11/xlib.pdf>.

[2] Xlib. (2018). Wikipedia, La enciclopedia libre. Fecha de consulta: 23:33, mayo 7, 2019 desde https://es.wikipedia.org/w/index.php?title=Xlib&oldid=108193359.

[3] Página web del proyecto XOSD. Disponible en <https://sourceforge.net/projects/libxosd/>.

[4] *Xlib − C Language X Interface*. Disponible en <https://www.x.org/releases/X11R7.5/doc/man/man3/XcreateFontCursor.3.html>.

[5] Notification Development Guidelines. Disponible en <https://wiki.ubuntu.com/NotificationDevelopmentGuidelines>.

[6] incBrain. (2016). How to use notify-send with C++? Ask Ubuntu. Disponible en <https://askubuntu.com/questions/730050/how-to-use-notify-send-with-c>.