Alumno: Daniel Fernández Núñez

Versión final.

2. El sistema de ventanas X: X.org

¿Qué es X-window?

El sistema de ventanas X (X Window System, X-Window o simplemente las X) se trata de la herramienta de software para el desarrollo de interfaces gráficas de usuario (GUI's) para estaciones de trabajo. Una GUI es en pocas palabras una interfaz usuario / computadora que se ejecuta en modo gráfico. X-Window para linux y para todos los sistemas más basados es UNIX lo que MS Windows es para los sistemas basados en DOS. Con una gran diferencia, que X-Window es un estándar para los sistemas de ventanas basados en UNIX. Esta estandarización supone que cualquier interfaz GUI puede ser ejecutada en cualquier computadora e incluso en varias a la vez.

El nombre de X se debe a que es un sucesor directo de un antiguo sistema de ventanas llamado W, ya que X es la letra posterior en un orden alfabético a W, el cual corría sobre el sistema operativo V (1980). El sistema de ventanas X surgió de la necesidad del MIT (Instituto Tecnológico de Massachusetts) de un sistema gráfico independiente de la plataforma usada. En 1983 se realizó un port de W a Unix y en 1984 se reemplazó el protocolo síncrono del W por otro asíncrono y las listas de displays por una nueva forma más dependiente de hardware para gestionar los gráficos y así nació la primera versión de X. X se convirtió así en el primer sistema de ventanas en ofrecer independencia del tipo de hardware y de los fabricantes.

Arquitectura Cliente / Servidor

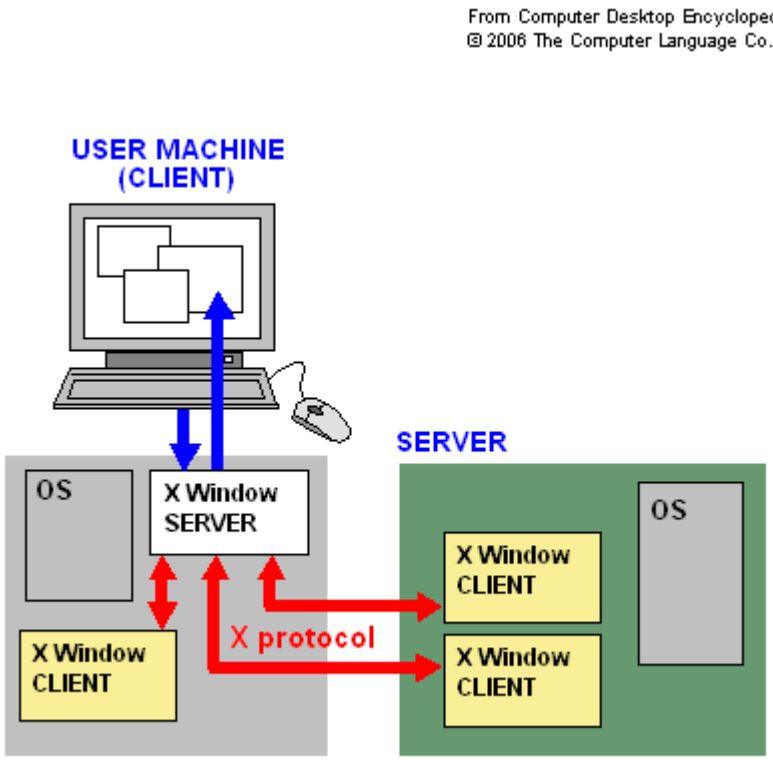
En Linux el proceso gráfico no es más que otro proceso que ejecuta el sistema operativo. Esto evita muchos problemas de estabilidad al kernel. Otra ventaja que tiene es la absoluta independencia del sistema operativo y el entorno gráfico. En contrapartida a todas estas ventajas, existe el inconveniente que el entorno gráfico reduce su velocidad en comparación a otros sistemas gráficos. Estos últimos incluyen los procesos referentes al subapartado gráfico en el propio núcleo, aunque esta práctica tiene la ventaja de que el sistema gráfico es más veloz. Se hace un gasto innecesario de recursos, aun sin usar ninguna aplicación, y existe el gran problema de que son más propensos a fallos del sistema debidos a errores en el apartado gráfico.

Toda la filosofía de X-Window se basa en la arquitectura cliente/servidor. Esta arquitectura es el modelo de sistema X mediante la cual los clientes, programas de aplicaciones, se comunican

con los servidores que controlan parte del hardware.

El programa que habilita un entorno gráfico X-Window en un ordenador es el servidor X (X server). Se le llama servidor ya que este programa sólo se dedica a escribir en pantalla lineas, cuadros y funciones gráficas básicas. El servidor X ofrece funciones gráficas primarias a las aplicaciones (clientes) que las soliciten y este las muestra en pantalla. El servidor es el programa encargado de gestionar un display. Un display se debe entender como la unidad formada por la o las pantallas y por los dispositivos de entrada, bien sea un ratón, un teclado, un trackball etc. todo este conjunto es un display. Un servidor puede servir a varios clientes a la vez. La otra parte de la arquitectura cliente / servidor es el cliente, que básicamente es una aplicación que se esta ejecutando en modo gráfico. El servidor simplemente es la unidad de visualización, que puede a su vez estar formada por varios monitores o pantallas físicas.

El servidor se encarga de captar las entradas del usuario y se las pasa a las aplicaciones o clientes X, dicha información proviene de los dispositivos de entrada del display, para que los clientes actúen en consecuencia. Los clientes tienen que captar esta información y operar. La respuesta del cliente es mandada al servidor ordenándole que dibuje dicha respuesta en la pantalla o las pantallas del display. Descodifican los mensajes de los clientes, como las peticiones de información o el movimiento de una ventana. Toda la comunicación entre el cliente y el servidor se realiza en lenguaje formal X window.



Esquema general del sistema de ventanas X.

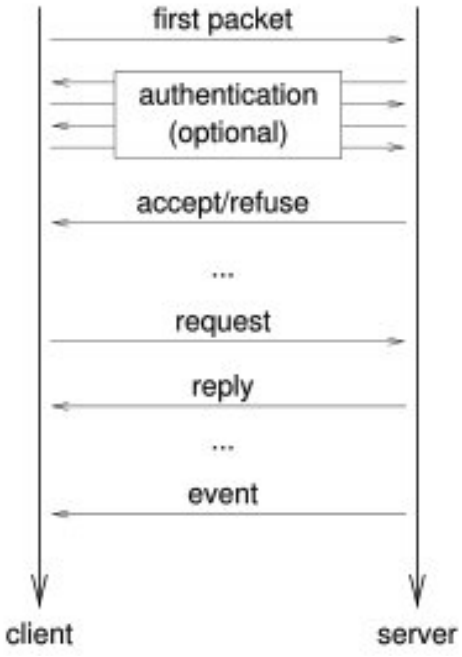
Todo esta arquitectura se debe a que el sistema X windows tiene una gran flexibilidad de uso en redes. La conexión del servidor X a los clientes no esta limitada a la misma máquina, sino que cualquier aplicación o cliente que se este ejecutando en una red se puede conectar a cualquier servidor. Por ejemplo, una forma de aprovechar esta flexibilidad es utilizar desde casa un ordenador personal conectado a una gran computadora para aprovechar la potencia de esa computadora desde casa. Esta práctica de compartir aplicaciones esta muy extendida, de hecho existen un tipo de ordenadores cuya única función es ejecutar un servidor X, se denominan "terminales X".

La comunicación entre el cliente y el servidor sistema X-Window se realizan mediante el

denominado **protocolo X** (X protocol), intercambiando paquetes de datos a traves de un canal. Tras la conexión entre servidor y cliente, donde este último es el que inicia la comunicación enviando su versión del protocolo y el tipo de autenticación que necesita, existiran cuatro tipos de paquetes que se intercambiaran entre servidor y cliente a lo largo de la conexión:

- Request: solicitud, el cliente solicita información del servidor o al mismo servidor para realizar una acción determinada.
- Reply: respuesta, el servidor responde a una petición. No todas las peticiones necesariamente van a generar una respuesta.
- Event: evento o suceso, el servidor informa al cliente de la ocurrencia de un evento (por ejemplo la pulsación de una tecla del teclado, movimiento o reescalado de la ventana, minimización).
- Error: el cliente envia un paquete de error si una petición es inválida. Ya que las peticiones son encoladas por el servidor, los paquetes de error generados por una petición pueden no ser enviados inmediatamente.

Esta comunicación tiene otras características: los paquetes tienen distinto tamaño según el tipo de paquete (los Request y Reply son de tamaño variable, mientras que los Event y los Error tienen el tamaño fijo de 32 bytes), los paquetes Request son numerados secuencialmente por el servidor según su orden de llegada a este,



Un ejemplo de una comunicación entre un cliente y un servidor.

El problema es que, como hemos podido ver en la descripción del protocolo, la programación con este lenguaje a tan bajo nivel es extremadamente complicada y laboriosa.Así a grandes rasgos podríamos comparar el protocolo X al lenguaje máquina.

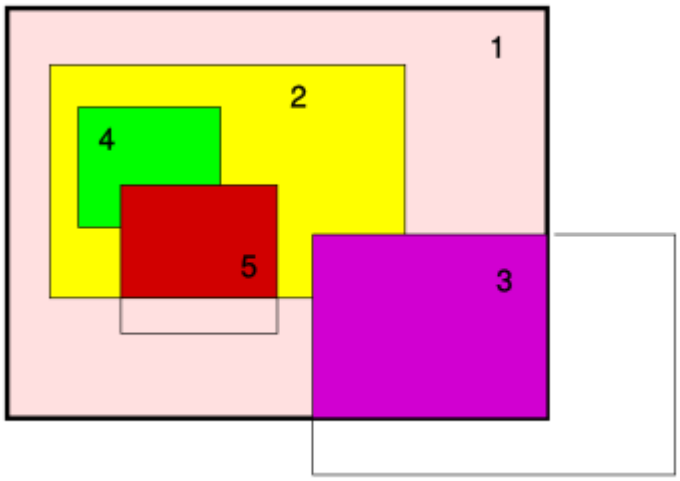
De la misma manera que el ensamblador proporciona potencia a la programación en lenguaje máquina, las funciones Xlib proporcionan la potencia del protocolo X con un coste menor. Xlib es un biblioteca de unas 300 funciones escritas en el lenguaje C que generan protocolo X, que facilitan la programación básica. Las funciones Xlib son el punto de partida para aprender a manejar X-Window y aunque sea imprescindible dominarlas, para programar a más alto nivel nos harán falta otras herramientas, como GTK+ o Qt. Al final de este tema, habrá un ejemplo de cómo usar las librerias Xlib para crear una sencilla aplicación X.

Ahora bien, ¿cómo tipo de jerarquia sigue X-Window para dibujar las ventanas en el display? Lo que normalmente nosotros llamamos una ventana en la mayoría de interfaces gráficas se denomina ventana *top-level* o de nivel superior. También podríamos entender como ventanas a las que están dentro de otras ventanas, esto es, subventanas de una ventana padre. Ejemplos de subventanas podrían ser los botones, menús, iconos, barras de scroll vertical, etc.

Un cliente puede solicitar la creación de una ventana, que de una forma más precisa se dice que solicita la creación de una subventana en una ventana existente. Como resultado, las ventanas creadas por clientes son colocadas en forma de arbol, con una cierta jerarquía. La raíz de ese árbol es lo que llamamos ventana raíz o *root window*, que es una ventana especial que crea automaticamente el servidor al comienzo de la sesión. Consecuentemente, todas las ventanas que creamos desde el inicio de la sesión van a *ser* directa o indirectamente subventanas de esta ventana raíz. En términos de visibilidad la ventana raíz es tan grande como la pantalla, y siempre se situa detras de el resto de ventanas.

No se garantiza que el contenido de una ventana no cambie en un futuro. Por ejemplo, el contenido de la ventana se perderá cuando la ventana se mueva, se maximice, se cubra por otra ventana parcial o totalmente. Se le puede solicitar al servidor X que guarde el contenido de la ventana, pero el servidor no está obligado a hacerlo. Por lo tanto, el cliente debe presuponer que el servidor no va a guardar el contenido, por lo que es responsabilidad del cliente tener que redibujar su contenido si existe algún cambio.

Cada ventana tiene asociadas ciertas caractaristicas o atributos, como su geometría (tamaño y posición), la imagen de fondo, etc. El protocolo incluye las peticiones referentes a la inspección o el cambio de estos atributos en una ventana. En X-Window las ventanas pueden ser InputOutput o InputOnly, donde las del primer tipo pueden ser mostradas en pantalla y ser usadas para dibujar en ellas, mientras que las del segundo tipo nunca se muestran en pantalla y solo sirven para recibir información.



Posicionamiente de ventanas: 1 es la ventana raíz, que cubre toda la ventana. 2 y 3 son ventanas de nivel superior; 4 y 5 son subventanas de 2; y la parte de 3 que cae fuera de 1 no se visualiza.

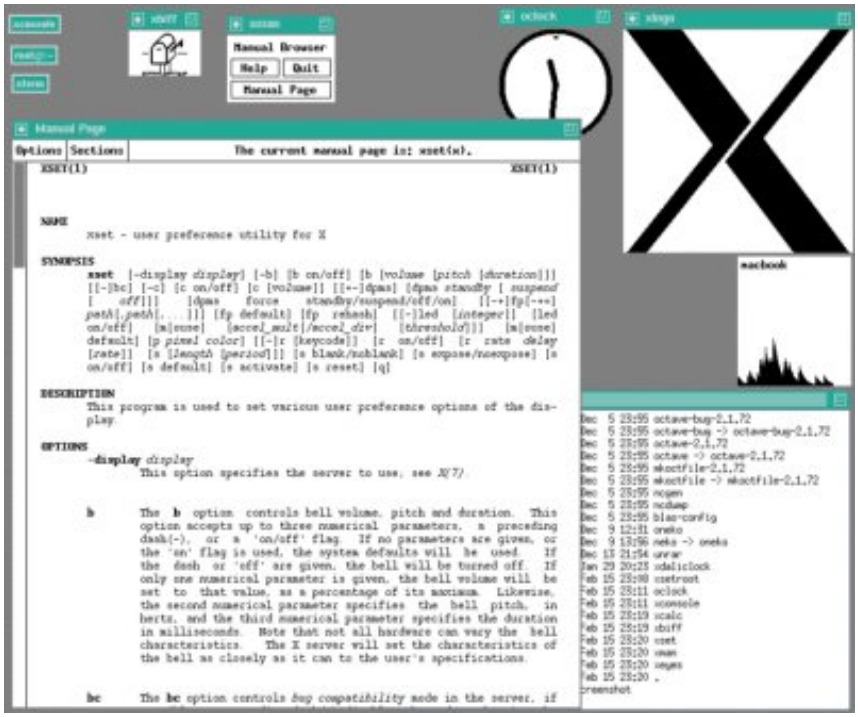
La parte decorativa del exterior de las ventanas y la barra del título son creadas por el gestor de ventanas, del cual se hablará con más profundidad en el siguiente tema. Aún así, es interesante señalar en este tema la existencia de gestores de ventana de re-parenting, que lo que hacen es que cuando se crea una ventana de nivel superior, este gestor cambia su parentesco con las otras ventanas y la convierte en una nueva ventana a la que gracias a esto le puede añadir las partes decorativas de las que hablamos antes. Aún así, el gestor de ventanas sigue siendo a efectos de la comunicación con el servidor X un cliente X cualquiera.

X.org

Como vimos antes, ya tenemos el sistema y el protocolo X. Entonces ahora lo que

necesitamos para poner todo esto en marcha es una implementación de este sistema. De entre varias existentes destacaremos a **X.Org**, ya que es una implementación de código abierto del sistema X Window System, que surgió como bifurcacion de proyecto Xfree86, y es una de las más usadas en la actualidad.

Este proyecto está a cargo de la X.Org Foundation, creada en 2004 cuando los desarrolladores que ofrecieron una implementación libre y estándar del servidor X unieron esfuerzos que antiguos desarrolladores de Xfree86, otra implementación del servidor X. El proyecto está alojado en freedesktop.org.



Antigua captura de una sesión de X.org usando el gestor de ventanas twm.

La primera versión del servidor X.org (X11R6.7.0), partió del código de Xfree86 4.4 RC2, debido a un cambio de licencia producido en este último en Febrero de 2004 (anteriormente se distribuía bajo la Licencia MIT y la nueva licencia generó mucha controversia por incompatibilidades con la GPL). En conflicto resultó en que varios de los anteriores desarrolladores de XFree86 se sumaron al proyecto, ya que se gestiona de una forma más abierta que Xfree86.

El servidor X.Org es muy popular entre los sistemas operativos libre basados en Unix, siendo adoptado por la gran mayoria de distribuciones de Linux y de las variantes de BSD. También ha sido adoptado por el sistema operativo de Sun, Solaris, aunque también incluyen en el paquete a Xsun, servidor propietario de Sun para arquitecturas SPARC. Versiones anteriores al Mac OS X 10.5 Leopard de Apple estaban basadas en Xfree86 hasta esta última versión que es una variante de X.Org.

DRI

El avance tecnológico ofrecía cada vez mejor hardware, y en el caso del harware gráfico no fue menos, por lo que era necesario hacer algo para aprovechar la potencia que nos ofrecía nuestra tarjeta gráfica pero sin renunciar a nuestro X Window System. Para ello nació el **DRI** (Direct Rendering Infraestructure, *Infraestructura de Renderizado Directo*), que es una interfaz usada en el X Window

System para que las aplicaciones de usuario puedan acceder de manera segura al hardware de video sin tener que pasar los datos por el servidor X, cosa que degrada el rendimiento. La aplicación principal es proporcionar aceleración por hardware a la librería Mesa, que es una implementación libre de OpenGL. DRI también ha sido adaptado para proveer aceleración OpenGL al framebuffer de Linux, sin ejecución de un servidor X.

El soporte OpenGL de DRI se realiza mediante la unión de varios componentes:

El primero de ellos es Direct Rendering Manager (DRM, Gestor de renderizado directo), que consiste en dos módulos del kernel: un módulo genérico llamado drm, y otro específico del chip gráfico que provee APIs para acceder a las diferentes clases de hardware gráfico (como pueden ser ATI o Nvidia). Algunas de las principales caracteristicas que DRM aporta a DRI son las siguientes:

- DRM se encarga del acceso sincrono al hardware gráfico. El sistema de renderizado directo tiene varias entidades (como por ejemplo el servidor X, el núcleo del sistema o kernel, clientes / aplicaciones que usan rendering directo) que compiten por el acceso directo al hardware gráfico. Para realizar el acceso sincrono, DRM ofrece acceso exclusivo al hardware (hardware lock) a cada componente. El uso exclusivo del hardware podría requerirse cuando el servidor X realiza rederizado en 2D, cuando una aplicación de render directo esta realizando una recuperación tras un error donde tiene que leer o escribir en el framebuffer, o cuando el kernel está asignando buffers de DMA.
- DRM refuerza de la política de acceso seguro al hardware gráfico del DRI. El servidor X, que se ejecuta como usuario root, obtiene acceso al framebuffer y a MMIO (Memoria mapeada para entrada y salida) del hardware gráfico a traves de /dev/mem. Los clientes que requieren de rendering directo no van a ser ejecutados como root, pero aún así van a necesitar similares recursos a los que accede el servidor X a través de /dev/mem. Entonces aquí es donde entra DRM, que ofrece a los clientes unos mapeados de memoria similares a los que se pueden obtener con /dev/mem pero con una serie de restricciones para no crear conflictos con el servidor X y otros recursos compartidos.
- DRM ofrece un motor genérico de DMA (Direct Memory Access, Acceso directo a memoria) para una mejor gestión de recursos del hardware gráfico.

El segundo componente de los mencionados antes es un driver para el espacio de usuario (userspace) , el cual contiene un driver OpenGL que típicamente realiza la labor de preparar buffers de comandos para ser enviados al hardware gráfico vía DRM, e interactúa con el sistema de ventanas para sincronizar el acceso al hardware de vídeo.

El tercer componente es un servidor, que por ejemplo puede ser en nuestro caso las X con la librería **libdri.so** y un driver DRI DDX para 2D.

El DRI fue inicialmente desarrollado por la compañía Precision Insight en cooperación con las compañías Red Hat y SGI. Tras la unión de Precision Insight con VA Linux y la salida de VA Linux de mundo de Linux, el DRI pasó a estar bajo el mantenimiento de Tungsten Graphics, una

nueva compañía formada por algunos de los desarrolladores que participaron en la creación de DRI de antigua Precision Insight. Actualmente Tungsten Graphics sigue siendo la principal encargada del desarrollo de DRI y muchos desarrolladores de código abierto siguen contribuyendo al proyecto a través del DRI project, alojado en freedesktop.org.

Ejemplo

Como ejemplo en este capítulo sobre el servidor X usaremos la librería Xlib para hacer un pequeño programa que nos muestre en una ventana un texto con alguna información de nuestra pantalla.

Como ya se dijo antes, las Xlib son un conjunto de funciones y macros realizadas en el lenguaje de programación C que especifican una interfaz de programación de bajo nivel para X. Xlib se basa en una filosofía de **eventos**. Un evento o mensaje es un flujo de datos generados de forma asincrona por el servidor X como resultado de actividad en un dispositivo o de efectos laterales de las propias funciones de Xlib. Un evento puede ser la pulsación de una tecla, el movimiento del ratón...).

Antes de entrar en materia, vamos a revisar algunos conceptos para la creación de una aplicación X.

En la siguiente lista se enumeran los pasos que se deberían seguir a la hora de programar:

1. Conectar con el servidor X.
2. Crear una ventana.
3. Darle indicaciones al gestor de ventanas, es decir, cómo debe el gestor de ventanas mostrar la ventana (en terminos de tamaño, posición, etcétera).
4. Decidir que eventos queremos que la ventana considere.
5. Crear un contexto gráfico para la salida.
6. Mostrar la ventana.
7. Introducir el bucle de eventos (generalmente un `while(1)`, que es donde la aplicación va a interactuar con el usuario y responder a los eventos. El programa recorrerá el bucle una y otra vez hasta que se salga de la aplicación (por medio de un `return` o matando el proceso).

Las funciones que se van a usar con Xlib las podemos agrupar en tres grupos:

1. Operaciones para la conexión con el servidor: `XOpenDisplay`, `XCloseDisplay`, ...
2. Solicitudes al servidor, que se subdividen en solicitudes de operaciones (`XCreateWindow`, `XCreateGC`,...) y solicitudes de información (`XGetWindowProperty`, ...).

3. Operaciones que locales al cliente: operaciones en la cola de eventos (`XNextEvent`, `XPeekEvent`, ...) y otras operaciones de datos locales (`XLookupKeysym`, `XParseGeometry`, `XSetRegion`, `XCreateImage`, `XSaveContext`, ...)

Xlib es una librería que hace poco más que ocultarnos los detalles de la implementación de la comunicación entre cliente y servidor. Incluso un simple programa programado solo con Xlib nos daría como resultado un montón de lineas de código.

Sin ir más lejos, el ejemplo es una muestra de la complejidad de Xlib: un sencilla ventana que nos saludará y nos mostrará información de nuestra pantalla, usando solamente las librerias de Xlib.

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(argc, argv)
     int argc;
     char **argv;
{
Display *d;
unsigned int
width,height,display_width,display_height,window_width,window_heigh
t;
int s;
int x = 0;
int y = 0;
Window w;
char *window_name = "Hola mundo!";
char *icon_name = "icono";
char cd_height[50], cd_width[50], cd_depth[80];
XEvent e;
XSizeHints size_hints;

/* Abrimos una conexión con el servidor X */
d=XOpenDisplay(NULL);
if(d==NULL) {
     printf("No se puede abrir un display\n");
     exit(1);
}

s=DefaultScreen(d);
```

```
/* Obtenemos el tamaño de la pantalla */
display_width = DisplayWidth(d,s);
display_height = DisplayHeight(d,s);

/* Establecemos cómo de grande será la ventana en proporción al
display */

window_width = display_width/3;
window_height = display_height/4;

/* Creamos una ventana */
w=XCreateSimpleWindow(d, RootWindow(d, s), x, y, window_width,
window_height, 1,
                              BlackPixel(d, s), WhitePixel(d, s));

/* Creamos ciertas restricciones referentes al tamaño de nuestra
ventana, */
size_hints.flags = PPosition | PSize | PMinSize;
size_hints.x = x;
size_hints.y = y;
size_hints.width = window_width;
size_hints.height = window_height;
size_hints.min_width = 200;
size_hints.min_height = 200;

/* La siguiente función es útil para decirle al gestor de ventanas
cómo
queremos que muestre nuestra ventana  */

XSetStandardProperties(d, w, window_name, icon_name, None, argv,
argc, &size_hints);

/* Selecionamos los eventos que queremos que la ventana tenga en
cuenta */
XSelectInput(d, w, ExposureMask | KeyPressMask);

/* Mostramos la ventana */
XMapWindow(d, w);

/* Bucle de eventos */
while(1) {
     XNextEvent(d, &e);
     /* Dibujamos o redibujamos la ventana */
     if(e.type==Expose) {
          /* Dibujamos unos rectangulos */
          height = window_height/2;
          width = 3 * window_width/4;
```

```
          x = window_width/2 - width/2;
          y = window_height/2 - height/2;

          XDrawRectangle(d, w, DefaultGC(d, s), x, y, width,
height);
          XDrawRectangle(d, w, DefaultGC(d, s), x-5, y-5, width+10,
height+10);
          XDrawRectangle(d, w, DefaultGC(d, s), x-10, y-10, width
+20, height+20);
          /* Dibujamos el texto */
          sprintf(cd_height,"    Alto: %d pixels",
DisplayHeight(d,s));
          sprintf(cd_width,"    Ancho: %d pixels",
DisplayWidth(d,s));
          sprintf(cd_depth,"    Profundidad de la ventana: %d
planos", DefaultDepth(d,s));
          XDrawString(d, w, DefaultGC(d, s), window_width/6,
window_height/3+10, "Hola, este es un ejemplo del uso de las
Xlib.",
                      strlen("Hola, este es un ejemplo del uso de las
Xlib."));
          XDrawString(d, w, DefaultGC(d, s), window_width/6,
window_height/3+10+15, "Tu pantalla tiene las siguientes
caracteristicas:",
                      strlen("Tu pantalla tiene las siguientes
caracteristicas:"));
          XDrawString(d, w, DefaultGC(d, s), window_width/6,
window_height/3+10+30, cd_height,
                      strlen(cd_height));
          XDrawString(d, w, DefaultGC(d, s), window_width/6,
window_height/3+10+40, cd_width,
                      strlen(cd_width));
          XDrawString(d, w, DefaultGC(d, s), window_width/6,
window_height/3+10+50, cd_depth,
                      strlen(cd_depth));
     }
     /* Salimos del programa cuando pulsamos una tecla */
     if(e.type==KeyPress)
     break;
}

     /* Cerramos la conexión con el servidor */
     XCloseDisplay(d);

     return 0;
}
```

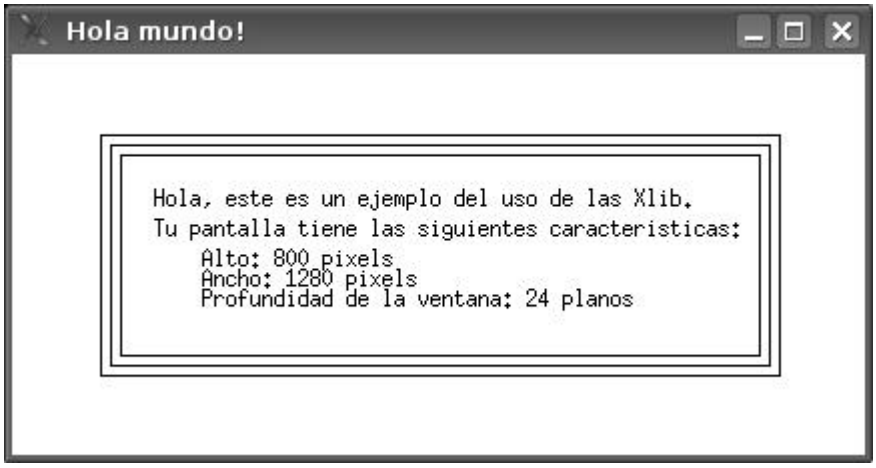
Podeis acceder al fichero de fuente en este enlace: [enlace al holamundo.c]

Para compilar el código hemos usado gcc con los siguientes parámetros (es importante indicarle el directorio en donde se encuentra Xlib, hemos puesto X11R6 porque es lo más usual en una distribución actual tipo Ubuntu o Debian):

```
gcc -o holamundo holamundo.c -Wall -L/usr/X11R6/lib -lX11
```

siendo `holamundo.c` el archivo fuente.

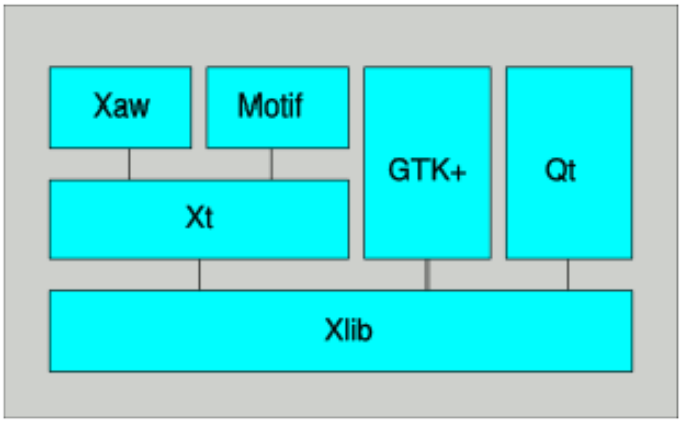
El resultado final de nuestro código sería la siguiente ventana:



Como podemos ver, las opciones que nos ofrece Xlib a priori son bastante limitadas, ya que no podemos colocar botones, menús o barras de desplazamiento. Estos widgets nos lo ofrecen otras librerias de más alto nivel, que a su vez por debajo usan Xlib. Hay dos tipos de estas librerias:

- Las que se apoyan en la librería Intrinsics (abreviada Xt) que ofrece soporte para widgets, aunque el propio conjunto de widgets (widgets set) especificos los ofrecen las librerias que usan Xt, como pueden ser Motif o Xaw (de este último hay un ejemplo en otro trabajo de la asignatura: http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Interfaces/enlightment/x_4.html)
- Las que ofrecen widget sets usando Xlib directamente sin pasar por Xt, como pueden ser los famosos toolkits GTK+ y Qt.

En el esquema siguiente podemos ver graficamente como se dividen las anteriores librerias:



Para más información de programación usando las Xlib, recomendamos estas webs:

- http://tronche.com/gui/x/xlib/ : completo manual que explica desde cero como usar las librerias.
- http://tronche.com/gui/x/xlib/function-index.html : documentación sobre las distintas funciones de Xlib.

Y para más ejemplos:

- http://www.visi.com/~grante/Xtut/basicwn.html

Bibliografía:

- http://wikipedia.org/
- http://www.netpecos.org/docs/linux/indice.html
- http://dri.freedesktop.org/wiki/
- http://xorg.freedesktop.org/wiki/
- http://www.paulgriffiths.net/program/c/hellox.php