UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Generación de eventos sintéticos de teclado y ratón en GNU/Linux con Xlib

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

1 Resumen de las ideas clave

Habitualmente, las aplicaciones solo esperan recibir eventos de entrada como resultado de la interacción con el usuario a partir de los medios tradicionales: teclado y ratón. Cuando estos dispositivos son utilizados, se generan eventos hacia el sistema operativo, quien los deriva al gestor de ventanas y este los dirige a la aplicación que corresponda.

Hay situaciones que hacen necesaria la generación de estos eventos para simular uno de estos dispositivos. Así sucede cuando el usuario no dispone de un ratón sino de una superficie táctil y se quieren habilitar sobre ella el reconocimiento de una serie de "gestos" o acciones con los dedos para simular un ratón físico e, incluso, aumentar el número de acciones que este puede realizar (como los denominados *trackpad* o t*ouchpad*). Y, más recientemente, los sistemas de realidad aumentada y realidad virtual.

En otras ocasiones es necesario enviar pulsaciones de teclas en teclados virtuales (también llamados *on-screen virtual keyboards*) que se utilizan para su uso en dispositivos táctiles donde no existe uno real conectado, para reemplazar los reales por avería, por dificultad de uso de uno real por cuestiones de discapacidad o, como es habitual en páginas web de bancos, como protección frente a dispositivos físicos que capturan las teclas pulsadas; con opción a variar incluso la geometría y/o la disposición de las teclas.

2 Objetivos

Hablaremos en este artículo de **cómo generar eventos de manera sintética** hacia el sistema y trataremos, mínimamente, los detalles de los posibles eventos y parámetros. Con ello obtendremos puntos de enlace a partir de los que buscar más funcionalidades y opciones de las mismas.

El **gestor de ventanas se comunica con las aplicaciones de usuario** por medio de eventos. El presente documento está encaminado a ofrecer una perspectiva inicial de cómo abordar esta comunicación a través de las funciones estandarizadas en su **interfaz de programación de aplicaciones** (*Application Programming Interface* o, abreviadamente, *API*). Lo haremos desde la plataforma GNU/Linux, como ejemplo práctico de *X Window System* (también llamado *las X*[1] o *X11*), que es el servidor gráfico de *Unix* (utilizado con variantes en *GNU/Linux*, *macOS* o *Android*) y al que se puede acceder a través de la implementación del API que la biblioteca de funciones *Xlib* implementa.

Para ello nos centraremos en los siguientes subojetivos:

- Enunciar la arquitectura del servidor gráfico de *Unix* y la parte del API que *Xlib* implementa para la comunicación de eventos entre aplicaciones.
- Enunciar los elementos básicos que es necesario añadir a una aplicación desarrollada en lenguaje C que utilice el API de *Xlib* y cómo compilarla.
- Reconocer los elementos de un ejemplo básico que Implemente la interacción con el ratón o con el teclado mediante *Xlib*.

El lector podrá experimentar, en este artículo, cómo **generar nuevos eventos desde una aplicación**. Se construirá una **emulación del ratón o teclado** a través de los mecanismos implícitos que un sistema de ventanas ofrece para inyectar, de forma artificial, nuevos eventos en la cola del gestor.

---

1 Y pronunciado habitualmente como si fuera un plural: "las Xs".

3 Introducción

Vamos a revisar la forma en que son generados y gestionados los eventos. En computadores, un sistema con interfaz gráfico (*graphical user interface*, GUI) es aquel [1] que ofrece la gestión de la pantalla en áreas de diferente geometría o ventanas, proporcionando un paradigma de interacción basado en la metáfora del escritorio [2]. La fig. 1a muestra un sistema[2] gráfico típico de entorno basado en UNIX, que utiliza el protocolo X11 para comunicar a dos componentes: el servidor (*display server*) y el gestor o administrador de ventanas (*window manager*). En otras plataformas no existe esta separación entre sus componentes: en *Windows* está el *Desktop Window Manager* y, en *macOS,* el *Quartz Compositor*.

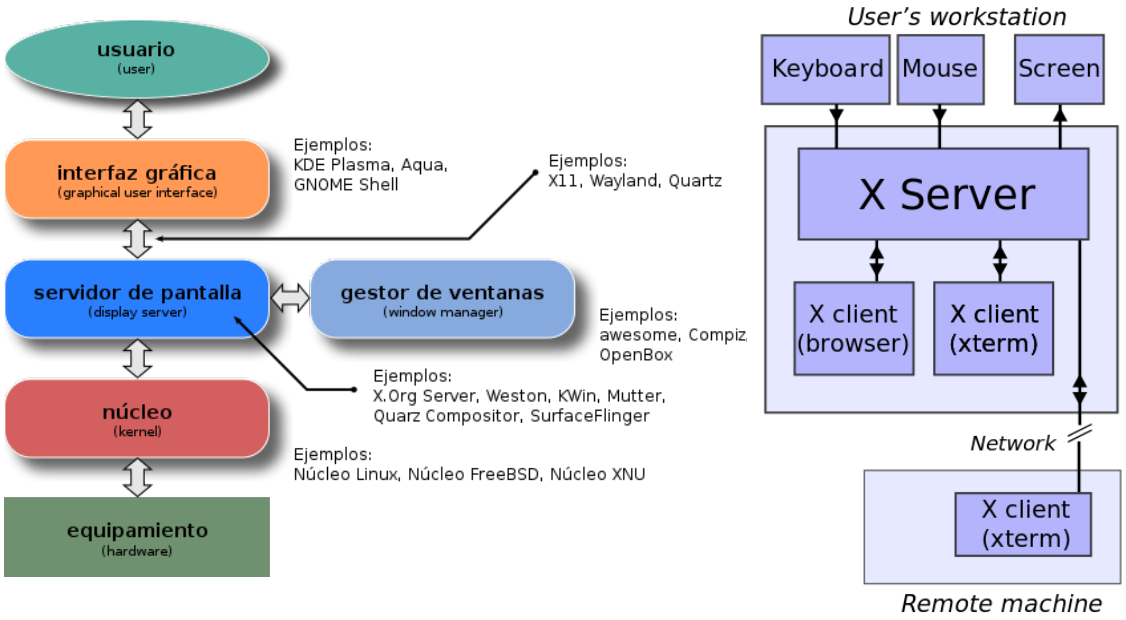

Figura 1: (a) Esquema de las capas de la interfaz gráfica de usuario (X.Org Server), (b) El servidor de X permite la interacción de aplicaciones locales y remotas con los dispositivos (pantalla, teclado y ratón) locales.

El elemento principal es el **servidor de pantalla o servidor gráfico** (*display server*, *window server* o *compositor*). Cualquier aplicación, que tiene salida gráfica, es un cliente de este servidor. Entre ellos se comunican a través de un protocolo (*server protocol*). El servidor recibe los datos del interfaz de entrada del sistema directamente a través del núcleo y los envía al cliente correspondiente, manteniendo en pantalla la salida gráfica de los clientes. Por otro lado, el **administrador de ventanas o gestor de ventanas** (*window manager*) se encarga de la decoración (el borde, la barra de título, los iconos de esta, el menú de sistema, etc) y la gestión (maximizarlas, minimizarlas, replegarlas, etc.) de las ventanas.

Las *X* constituyen la definición de un servidor de pantalla que puede funcionar a través de la red. Así lo muestra la fig. 1b, en la que el servidor recibe entradas de teclado y ratón locales, mostrando en pantalla un navegador y un terminal que se ejecutan en local. Junto a estas aplicaciones, otro terminal se está ejecutando sobre una máquina remota. Este servidor proporciona [1] las operaciones de dibujo, creación de ventanas e interacción con el teclado y el ratón. *X* no es un gestor de ventanas. Esto permite al usuario instalar varios gestores de ventanas e intercambiarlos sin reiniciar el sistema.

2 Imagen extraída de <https://en.wikipedia.org/wiki/X.Org_Server>.

¿Quieres jugar un poco antes de empezar? Si estás en un sistema basado en Unix, prueba a ejecutar desde el terminal las órdenes `xdpyinfo` (que muestra información sobre el servidor X que tienes instalado), `xlsclients` (que ofrece un listado de la aplicaciones gráficas que están ejecutándose en este momento) y `xev` (que te mostrará la avalancha de eventos que recibe una aplicación del servidor. Para poder gestionar toda esta información desde las aplicaciones, se utiliza el API de funciones que ofrece el protocolo X11 [3]. Existen diferentes implementaciones de este protocolo que permite a las aplicaciones comunicarse con el servidor, véase fig. 2, como Xlib, XCB, GTK+, Qt, FLTK o Motif. La opción más portable es Xlib (véase [4] y [5]), que constituye una biblioteca de funciones de bajo nivel, escrita en C. En ella nos vamos a centrar.

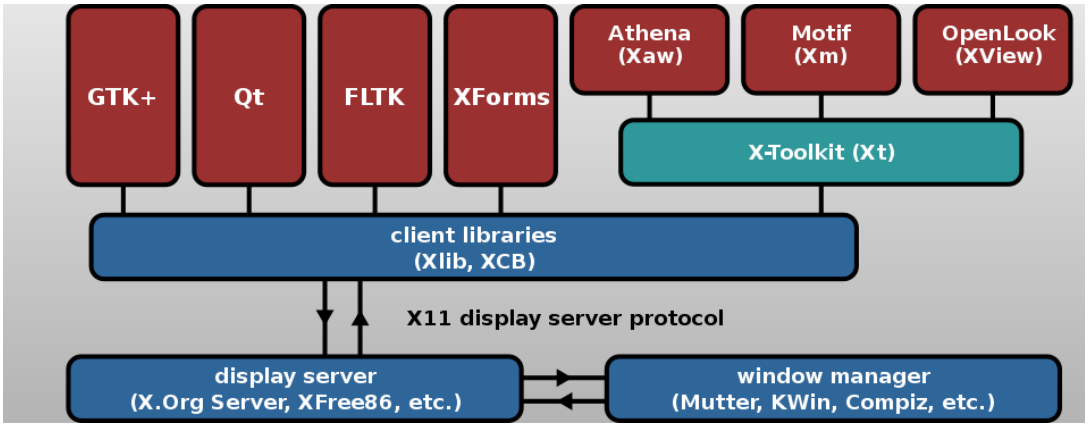

Figura 2: Las aplicaciones pueden usar diferentes bibliotecas para generar su interfaz gráfico [3].

4 Desarrollo

Ahora que hemos hablado de qué hay por debajo de las aplicaciones y de que se pueden enviar eventos a la cola del sistema, vamos a profundizar en ejemplos de código que nos permitan experimentar con los eventos. La primera aproximación es muy breve, pero sirve para ir centrando cosas: las estructuras de datos básicas, una mínima secuencia de operaciones, cómo se compila el ejecutable y las necesidades de instalación. Aquí fijaremos nuestra atención en la función *XwarpPointer*. Las otras dos servirán para ver casos de uso completos. En ambos casos veremos el uso de la función *XSendEvent*.

4.1 Un primer ejemplo de código

Siguiendo el ejemplo[3] de [6], véase el listado 1, vamos a ver cómo empezar a poner en una aplicación las funciones que nos permitirán mover el ratón por la pantalla. Para poder gestionar estas operaciones, una aplicación se estructura en base a: establecer una conexión con el servidor gráfico (que se ha de inicializar), a la que se le pueden enviar instrucciones o recibir datos y, que al terminar, deberá ser cerrada.

El *display* es el término que representa la abstracción del sistema gráfico de entrada y salida compuesto, básicamente, por la pantalla, el teclado y el ratón. En el *display* se realiza la interacción con el usuario de forma gráfica utilizando

3 Ha sido necesario revisar ruta de la cabecera que se incluye en la versión original, así como ampliar algunos mensajes de la ejecución para observar ciertos detalles .

la metáfora **WIMP** (este acrónimo está formado por las iniciales de los elementos que la componen "Windows, Icons, Menus, Pointer"[4]).

Para utilizar esta abstracción y las operaciones que se describen en el API sobre él, es necesaria la directiva de `include` para `Xlib.h` que se incluye al principio de la aplicación. Veamos ahora las tres operaciones a partir de la descripción de las mismas en la página del manual[5] de cada una. Para inicializar la conexión al servidor X se utiliza

```
Display *XopenDisplay( char *display_name );
```

donde *`display_name`* es una cadena de caracteres o *`NULL`*, en cuyo caso se toma el valor de la variable de entorno *`DISPLAY`*. La cadena de caracteres tiene el formato "hostname:number.screen_number", lo que permite establecer una conexión a través de la red con una máquina (*`hostname`*), uno de los posibles servidores gráficos en ejecución (*`number`*) y un posible escritorio (*`screen_number`*) virtual del servidor. La estructura `Display` contiene la información del servidor X e identifica la conexión que ha podido ser establecida tanto en remoto (p. ej. utilizando TCP), como en local.

Por su parte, para cerrar la conexión se utilizará

```
int XCloseDisplay(Display *display);
```

que, a partir del valor de *`display`*, liberará los recursos asignados.

```
#include <X11/Xlib.h>


int main() {
  int delta_x = 500,
      delta_y = 160;


  Display *display = XOpenDisplay(0);
  Window root = DefaultRootWindow(display);


 XWarpPointer(display, None, root, 0, 0, 0, 0,
              delta_x, delta_y);
  XCloseDisplay(display);
  return 0;
}
```

Listado 1: Ejemplo mínimo de generación de eventos de ratón.

Una posible instrucción que enviar al *display* es *XWarpPointer*. Es una función de `Xlib` que introduce en la cola de eventos del sistema gráfico los mismos eventos que hubiera generado el ratón físico, si se hubiera movido a las coordenadas especificadas. La función se define como:

```
int XWarpPointer(Display *display, Window src_w, Window dest_w,
          int src_x, int src_y, unsigned int src_width,
          unsigned int src_height, int dest_x, int dest_y);
```

4 Para contextualizar el término WIMP en computadores puede echar un vistazo en <https://en.wikipedia.org/wiki/WIMP_(computing)>.

5 Con la orden `man` o en línea en "X.Org Foundation", disponible en <https://www.x.org/wiki>.

siendo capaz de mover el puntero a las coordenadas (`dest_x, dest_y`). ¿Lo has probado? Pruébalo, cópialo en un fichero de nombre *`simulatingMouseClick.c`*, compílalo y ejecútalo con

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

¡¡Funciona!! Mueve el cursor a las coordenadas (500, 160), especificadas en el código. Pruébalo, asegúrate de mover el cursor y verás como siempre vuelve a esas mismas coordenadas.

4.2 Generación de eventos de ratón

En este apartado vemos un ejemplo[6] [7] interesante: lanza una ininterrumpida secuencia de clics, a razón de uno por segundo, sobre las coordenadas donde lleves al cursor con el ratón. Así, que se ve que se selecciona y se deselecciona una palabra, si el cursor están en un terminal o sobre una ventana de navegador. Y si lo dejas encima del icono de guardar de un editor que tengas abierto, verás como se realiza la función de guardar, puesto que le va a enviar el clic al icono.

Ahora, vamos a emular el evento de clic de ratón. Para ello, los listados 2 y 3 muestran cómo ampliar el ejemplo anterior con la función `mouseClick` [6]. Suponiendo que se guarda el código en un fichero de nombre `autoclick.c`, en un terminal se compila con la orden:

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

y se llama al ejecutable con dos valores para la columna y fila en la que se quiere posicionar al cursor:

```
$ autoclick 200 300
```

Este ejemplo utiliza las funciones de *Xlib*: *`XQueryPointer`* y *`XSendEvent`*, que se definen como:

```
Bool XQueryPointer(Display *display, Window w,
                   Window *root_return, Window *child_return,
                   int *root_x_return, int *root_y_return,
                   int *win_x_return, int *win_y_return,
                   unsigned int *mask_return);
Status XSendEvent(Display *display, Window w, Bool propagate,
                  long event_mask, XEvent *event_send);
```

La primera, *`XQueryPointer, sirve`* para obtener las coordenadas actuales del cursor.

La segunda, *`XSendEvent`*, es la que envía realmente el evento al servidor gráfico; para lo que necesitaba saber las coordenadas en que estaba el ratón donde se quiere simular el evento de clic.

Observa que se identifica el botón (`button`) que se simula que ha sido presionado. Y, también, que se envían dos eventos, uno de pulsación de botón del ratón y otro de "liberación" del botón. Esto es, que el usuario ha pulsado y soltado el botón. De no ser así se entendería que el usuario tiene el dedo apretando el botón y no lo suelta.

---

6 Al código original le faltaba la inclusión de `unistd` (para `usleep` y `sleep`).

```
#include<stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <unistd.h> // usleep, sleep

// Simulate mouse click
void click (Display *display, int button) {
  // Create and setting up the event
  XEvent event;
  memset (&event, 0, sizeof (event));
  event.xbutton.button = button;
  event.xbutton.same_screen = True;
  event.xbutton.subwindow = DefaultRootWindow (display);
  while (event.xbutton.subwindow)    {
      event.xbutton.window = event.xbutton.subwindow;
      XQueryPointer (display, event.xbutton.window,
           &event.xbutton.root, &event.xbutton.subwindow,
           &event.xbutton.x_root, &event.xbutton.y_root,
           &event.xbutton.x, &event.xbutton.y,
           &event.xbutton.state);
  }
  // Press
  event.type = ButtonPress;
  if (XSendEvent (display,
                 PointerWindow, // A la ventana que está bajo el puntero
                 True, ButtonPressMask, &event) == 0)
      fprintf (stderr, "Error to send the event!\n");
  XFlush (display);
  usleep (1);
  // Release
  event.type = ButtonRelease;
  if (XSendEvent (display, PointerWindow, True,
                   ButtonReleaseMask, &event) == 0)
       fprintf (stderr, "Error to send the event!\n");
  XFlush (display);
  usleep (1);
}
 ...
```

Listado 2: Ejemplo de generación de eventos de ratón (1º parte).

```
...
int main (int argc, char *argv[])
{
  int starting = 3, x = 0, y = 0;
  // Open X display
  Display *display = XOpenDisplay (NULL);
  if (display == NULL)     {
      fprintf (stderr, "Can't open display!\n"); return -1;
  }
  // Wait 3 seconds to start
  printf ("Starting in   "); fflush (stdout);
  while (starting > 0)     {
      printf ("\b\b\b %d...", starting); fflush (stdout);
      sleep (1); starting--;
  }
  printf ("\n");
  // Start
  while (1)    {
      click (display, Button1);
      sleep (1);
    }
  // Close X display and exit
  XCloseDisplay (display);
  return 0;
}
```

Listado 3: Ejemplo de generación de eventos de ratón (2ª parte).

4.3 Generación de eventos de teclado

Veamos ahora a partir del ejemplo de [8] cómo enviar eventos de teclado a una aplicación desde otra, de manera que la que los recibe los trata de la misma manera que si fueran generados por un teclado físico. En este caso, el ejemplo revisado simulará que el usuario ha pulsado en el teclado físico la tecla de la flecha del cursor hacia abajo. La ventana activa recibirá este evento y lo gestionará como si se hubiera presionado esa tecla. ¡Y será transparente para ella, no podrá diferenciar si lo ha generado o no un dispositivo físico de uno virtual!

¿Quieres probarlo? El código repartido entre los listados 4 y 5 muestra el ejemplo indicado con dos pequeñas modificaciones para poder generar el ejecutable con el compilador de *C*. Prueba con esta secuencia de órdenes desde el terminal, junto al archivo *XFakeKey.c* (o como tú lo quieras llamar):

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

```
$ sleep 3;  XFakeKey
```

La segunda orden te da tres segundos para que puedas llevar el cursor a la aplicación que quieras que reciba el evento. Utiliza una donde sea visible la acción de las flechas del cursor del teclado.

```
#include <X11/Xlib.h>
#include <X11/keysym.h>
#define true 1
#define false 0
#define bool int

// The key code to be sent. A full list of available codes can be found
in /usr/include/X11/keysymdef.h
#define KEYCODE XK_Down

// Function to create a keyboard event
XKeyEvent createKeyEvent(Display *display, Window win,
                         Window winRoot, bool press,
                         int keycode, int modifiers) {
   XKeyEvent event;

   event.display     = display;
   event.window      = win;event.root  = winRoot; event.subwindow = None;
   event.time        = CurrentTime;
   event.x           = 1; event.y           = 1;
   event.x_root      = 1; event.y_root      = 1;
   event.same_screen = True;
   event.keycode     = XKeysymToKeycode(display, keycode);
   event.state       = modifiers;

   if(press) event.type = KeyPress;
   else      event.type = KeyRelease;
   return event;
}
...
```

Listado 4: Ejemplo de generación de evento de teclado (1ª parte).

Observa que se envían dos eventos, uno de pulsación de tecla y otro de “liberación” de tecla. Esto es, que el usuario ha pulsado y soltado la tecla. De no ser así se entendería que el usuario tiene el dedo apretando la tecla y no la suelta … Para simular eso habría que enviar repetidos mensajes de “tecla pulsada”.

```
...
int main() {
 // Obtain the X11 display.
 Display *display = XOpenDisplay(0);
 if (display == NULL) return -1;

 // Get the root window for the current display.
 Window winRoot = XDefaultRootWindow(display);

 // Find the window which has the current keyboard focus.
   Window winFocus;
   int    revert;
   XGetInputFocus(display, &winFocus, &revert);

 // Send a fake key press event to the window.
 XKeyEvent event = createKeyEvent(display, winFocus, winRoot,
                                  true, KEYCODE, 0);
 XSendEvent(event.display, event.window, True, KeyPressMask,
            (XEvent *)&event);

 // Send a fake key release event to the window.
 event = createKeyEvent(display, winFocus, winRoot, false, KEYCODE, 0);
 XSendEvent(event.display, event.window, True, KeyPressMask,
              (XEvent *)&event);
 // Done.
 XCloseDisplay(display);
 return 0;
}
```

Listado 5: Ejemplo de generación de evento de teclado (2ª parte).

¿Y si quiero utilizar otra tecla? Como dice en el código, en */usr/include/X11/keysymdef.h* encontrarás la lista de códigos. Yo la he cambiado por la letra 'n' cambiando el valor de KEYCODE con:

```
#define KEYCODE XK_n
```

¿Y si quisiera enviar una cadena de caracteres? ¡No es trivial puesto que no son los ordinales de los caracteres! Son códigos que se han hecho corresponder con el estándar *Unicode* (ISO 10646). Están definidos en keysymdef.h y son parejas de símbolo y ordinal en Unicode, junto a una descripción del carácter en este estándar, como p. ej. para la letra 'a':

```
#define XK_a            0x0061  /* U+0061 LATIN SMALL LETTER A */
```

¡Pero se puede hacer! ¿Una pista? Prueba una cadena de caracteres que tenga dígitos, vocales sin tilde y las consonantes sin la 'ñ'.

5 Conclusión

Cuando hay que desarrollar una aplicación que interactúe con el sistema y con el resto de aplicaciones en ejecución puede que estás hayan sido pensadas para ello. En otros casos, las aplicaciones solo esperan recibir eventos como resultado de la interacción con el usuario a partir de los medios tradicionales: teclado y ratón. Aquí hemos expuesto qué se puede hacer cuando una de estas aplicaciones queremos que pueda ser receptora de un nuevo dispositivo de interacción que, p. ej., sustituya al ratón en un sistema de realidad virtual o aumentada. También es aplicable a una aplicación de visión por computador que ha detectado un gesto con el dedo, para que pueda hacer que se mueva el ratón del sistema hasta unas ciertas coordenadas o enviar una pulsación de una tecla para que pasen las páginas en un visor de ficheros PDF. Si estás pensando que este es un posible camino hacia la construcción de un teclado virtual … ¡estás en lo cierto! Y, también, podría ser la manera en que un mando remoto o una aplicación que "lea" gestos realizados con la mano (quizá letras pintadas en el aire con la mano) pueda enviar esas letras a una aplicación.

Este artículo ha explorado la ruta que siguen los eventos por el interior de un sistema operativo, tipo *Unix*, a través del gestor gráfico estándar *X Window System*. Se han expuesto ejemplos breves y reales, de uso en computadores sobre el sistema operativo *GNU/Linux,* utilizando *Xlib* para generar, desde una aplicación desarrollada en lenguaje C, eventos sintéticos de teclado y ratón.

Ha quedado fuera de la exploración y lo dejo para el lector interesado, ver cómo enviar una cadena de caracteres o que existen otras formas, diferentes de *Xlib*, de implementar esta conexión como *XCB* o que existen otros protocolos como *Wayland y otras bibliotecas como XTest*. Y que otros lenguajes, como Python, también ofrecen interfaces similares.

6 Bibliografía

[1] Wikipedia contributors. (2018). X Window System. In Wikipedia, The Free Encyclopedia. Disponible en <https://en.wikipedia.org/wiki/X_Window_System >.

[2] J. Hicks. (2013). *40 years of icons: the evolution of the modern computer interface. Diary of a WIMP at middle age*. Disponible en <https://www.theverge.com/2013/3/21/4127110/40-years-of-icons-the-evolution-of-the-modern-computer-interface>.

[3] A Brief intro to X11 Programming. Disponible en <http://math.msu.su/~vvb/2course/Borisenko/CppProjects/Gwindow/xintro.html>.

[4] X Window Programming/Xlib. Disponible en <https://en.wikibooks.org/wiki/X_Window_Programming/Xlib>.

[5] Xlib − C Language X Interface. Xwindow System Standard .X Version 11, Release 6.7. Disponible en <https://www.x.org/docs/X11/xlib.pdf>.

[6] Question Simulating a mouse click. Disponible en <https://www.linuxquestions.org/questions/programming-9/simulating-a-mouse-click-594576/>.

[7] Pioz. autoclick.c. Disponible en <https://gist.github.com/pioz/726474 >.

[8] Sending fake keypress events to an X11 window. Disponible en <http://www.doctort.org/adam/nerd-notes/x11-fake-keypress-event.html >.

[9] Sending X11 click event doesn't work with some windows. Disponible en <https://stackoverflow.com/questions/12701069/sending-x11-click-event-doesnt-work-with-some-windows>.