EN ESTE NÚMERO: CÓMO CONFIGURAR UN SERVIDOR DNS

PROGRAMACIÓN ACTUAL

www.prensatecnica.com

995 pts.
PORTUGAL 1250 ESC (CONT)

AÑO 2 • NÚMERO 23

NIVEL BÁSICO
Control de formularios y aplicaciones en Delphi

LAS EMPRESAS DEMANDAN
Herramientas de administración en Oracle
WINDOWS NT: Gestión de discos y sistemas de tolerancia de fallos

BAJO NIVEL
El formato VRML en Programación 3D

CAMPUS ACTUAL
INTERNET: Evolución de Protocolos: IPv4 versus IPv6
Todo lo que no sabías de las *cookies*

VISUAL BASIC 6.0
Todas las novedades de la nueva versión

EN EL CD-ROM

PROGRAMACIÓN
- Lotus eSuite DevPack 1.5 Trial
- JDK 1.2
- DadvinaFence
- MP3Enc.ocx y MP3Play.ocx
- Applet Marquee Wizard
- W32Dasm

HERRAMIENTAS INTERNET
- HotMetal Pro 5.0 Trial
- Navegador HotJava 1.1.5
- Internet Explorer 4.01 SP1
- Java VM para IE 4.x
- Netscape Navigator 4.5 PR
- mIRC 5.5

WINDOWS
- WinZip 7.0
- WinRar 2.05
- Acrobat Reader 3.02
- WinHacker95 2.02

Windows DNA
La arquitectura para desarrollo de aplicaciones distribuidas

Prens@ Técnic
00023
8 413042 087779

Autor: Santiago Romero

Nivel: Iniciación

Eventos en X-Window (II)

Para finalizar con el tema de eventos nos introduciremos en la enorme estructura XEvent y la gran cantidad de información relacionada con los eventos que aportará a nuestro programa, así como a las funciones utilizadas para la recuperación de la misma, todo ello completado por sencillos ejemplos.

El mes pasado se vio la manera de seleccionar los eventos que se deseaba que nuestra aplicación fuera capaz de recibir, pero sin entrar en la forma en que se reciben, detectan y procesan estos eventos, que es precisamente el objetivo del capítulo de este mes.

Como ya se comentó, los eventos enviados por el servidor se almacenan en una cola de entrada desde donde el cliente debe leerlos y procesarlos. El servidor envía al cliente una estructura de datos que contiene tanto el tipo de evento ocurrido, como otra información útil relativa a ese evento, como por ejemplo la posición del *mouse* si el evento era un *ButtonPress*, o la tecla pulsada ante un *KeyPress*.

Ante tal cantidad de eventos disponibles, se hacen necesarias muchas estructuras de datos (una que devuelva información ante una pulsación de teclado, otra para información sobre el ratón, etcétera), pero X Window las ha fundido todas en una mediante el operador *union*, de forma que todos los eventos envían la misma estructura al programa cliente, una estructura que contiene los datos de cualquier posible evento, de tal modo que es la aplicación la encargada de mirar el campo apropiado para recuperar esta información.

Explicado de una manera simplista, supongamos que se dispone de sólo dos eventos: pulsación de tecla y pulsación de botón del mouse. Serían necesarias dos estructuras distintas para dotar de información extra al cliente, algo similar a:

```
struct EventoTecla
{
 char tecla_pulsada;
};

struct EventoBoton;
{
 char boton;
 int x, y;
};
```

Lo que hace X-Window es crear una sola estructura como unión de ambas, y pasar al cliente siempre esa estructura (sea cual sea el evento, pues ya se encargará la aplicación de utilizar uno u otro campo):

```
struct Evento
{
 char QueEvento;   /* 0=teclado,
1=raton */
 EventoTecla datos_teclado;
 EventoBoton datos_boton;
}
```

EVENTOS Y ESTRUCTURAS

En la tabla 1 pueden observarse los nombres de los diferentes eventos y las estructuras de datos (definidas en *X11/Xlib.h*) que contienen la información de los mismos, antes de pasar a ver la estructura XEvent, que contendrá la información de todos los eventos como en el ejemplo anterior.

Si el lector todavía no ha entendido nada sobre el tema de

FIGURA 1. SALIDA DEL PROGRAMA TEST1.

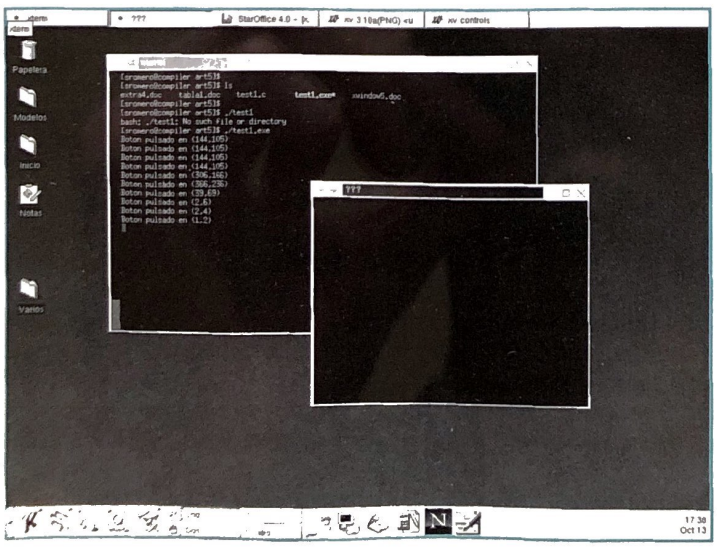


TABLA 1. EVENTOS Y ESTRUCTURAS

Evento	Estructura Utilizada	Campo utilizado
KeyPress	XKeyEvent	xkey
KeyRelease	XKeyEvent	xkey
KeymapNotify	XKeyMapEvent	xkeymap
ButtonPress	XButtonEvent	xbutton
ButtonRelease	XButtonEvent	xbutton
EnterNotify	XCrossingEvent	xcrossing
LeaveNotify	XCrossingEvent	xcrossing
MotionNotify	XMotionEvent	xmotion
CreateNotify	XCreateWindowEvent	xcreatewindow
DestroyNotify	XDestroyWindowEvent	xdestroywindow
GravityNotify	XGravityEvent	xgravity
UnmapNotify	XUnmapEvent	xunmap
MapNotify	XMapEvent	xmap
CirculateNotify	XCirculateEvent	xcirculate
ConfigureNotify	XConfigureEvent	xconfigure
ReparentNotify	XReparentEvent	xreparent
FocusIn	XFocusChangeEvent	xfocus
FocusOut	XFocusChangeEvent	xfocus
MapRequest	XMapRequestEvent	xmaprequest
ConfigureReq.	XConfigureRequestEvent	xconfigurerequest
ResizeRequest	XResizeRequestEvent	xresizerequest
CirculateReq.	XCirculateRequestEvent	xcirculaterequest
PropertyNotify	XPropertyEvent	xproperty
Expose	XExposeEvent	xexpose
ColormapNotify	XColormapEvent	xcolormap
SelectionClear	XSelectionClearEvent	xselectionclear
SelectionReq.	XSelectionRequestEvent	xselectionrequest
SelectionNotify	XSelectionEvent	xselection

LISTADO 1. EJEMPLO DE EVENTOS Y ESCRITURA EN PANTALLA

```
/* Ejemplo de eventos y escritura en pantalla */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

void main()
{
 Display *display;
 Window window;
 Font font;
 GC gc;
 XEvent evento;
 int pantalla;
 unsigned long negro, blanco;

 display = XOpenDisplay( NULL );
 font = XLoadFont( display, "8x16" );
 pantalla = DefaultScreen( display );
 negro = BlackPixel( display, pantalla );
 blanco = WhitePixel( display, pantalla );

 window = XCreateSimpleWindow( display,
        DefaultRootWindow( display ), 350, 0, 500, 400,
        1, negro, blanco );

 XSelectInput( display, window, ButtonPressMask | KeyPressMask );

 XMapWindow( display, window );
 gc = XCreateGC( display, window, 0, NULL );
 XSetFont( display, gc, font );

 while( 1 )
 {
  XNextEvent( display, &evento );
  switch( evento.type )
  {
   case ButtonPress:
    XDrawString( display, window, gc,
            evento.xbutton.x,
            evento.xbutton.y, "X", 1);
    break;

   case KeyPress:
    exit(0);
  }
 }
}
```

las estructuras y los eventos no debe preocuparse, pues aún es necesario mostrar y desarrollar la estructura XEvent para comprender todo el proceso.

ESTRUCTURA XEVENT

La estructura XEvent es el paquete de información que el servidor envía al cliente cuando ocurre el evento. Lo primero que haremos será ver el contenido de esta estructura (que lo relacionará directamente con las estructuras de eventos vistas en el anterior apartado), para después comentar la manera de extraer los eventos desde la cola de eventos (con el fin de trabajar con la estructura).
Veamos el contenido de la estructura XEvent:

```
typedef union _XEvent
{
 int type;
 XAnyEvent xany;
 XKeyEvent xkey;
 XButtonEvent xbutton;
 XMotionEvent xmotion;
 XCrossingEvent xcrossing;
 XFocusChangeEvent xfocus;
 XExposeEvent xexpose;
 XGraphicsExposeEvent
xgraphicsexpose;
 XNoExposeEvent xnoexpose;
 XVisibilityEvent xvisibility;
 XCreateWindowEvent
xcreatewindow;
 XDestroyWindowEvent
xdestroywindow;
 XUnmapEvent xunmap;
 XMapEvent xmap;
 XMapRequestEvent xmaprequest;
 XReparentEvent xreparent;
 XConfigureEvent xconfigure;
 XGravityEvent xgravity;
 XResizeRequestEvent xresizerequest;
 XConfigureRequestEvent
xconfigurerequest;
 XCirculateEvent xcirculate;
 XCirculateRequestEvent
xcirculaterequest;
 XPropertyEvent xproperty;
 XSelectionClearEvent xselectionclear;
 XSelectionRequestEvent
xselectionrequest;
 XSelectionEvent xselection;
 XColormapEvent xcolormap;
 XClientMessageEvent xclient;
 XMappingEvent xmapping;
 XErrorEvent xerror;
 XKeymapEvent xkeymap;
 long pad[24];
} XEvent;
```

Los eventos se almacenan en una cola de entrada desde donde el cliente debe leerlos

Antes de ver ejemplos sobre el tema, cabe decir que el bucle de mensajes deberá:

1.- Saber extraer la primera estructura XEvent disponible en la cola de eventos (recogiendo por tanto el primer evento de la cola).
2.- Saber identificar el evento ocurrido. Esto se hace, como veremos a continuación, mediante el campo *type* de la estructura.
3.- Dependiendo del tipo de evento, actuar en consecuencia y recoger de la estructura XEvent los datos necesarios a partir de sus diferentes campos. Si, por ejemplo, recibimos un evento KeyPress, accederemos a (como se verá en la definición de la estructura) *xevent.xkey* para informarnos acerca de dicha pulsación de teclado. De ahí que dentro de esta estructura XEvent estén definidas diferentes estructuras, útiles o no dependiendo del evento recibido.

El servidor envía al cliente una estructura de datos con información sobre el evento

Veamos en pseudocódigo nuestra aplicación X-Window:

```
Programa:
 Declarar variables.
 Inicializar display, ventanas, etc.
 Seleccionar eventos deseados
(XSelectInput)

 Bucle de Eventos:
  Coger el primer evento de la cola.
   Si evento.type es KeyPress
 Leer tecla de xevent.xkey.keycode
   Si evento.type es ButtonPress
     Leer datos de
 xevent.xbutton.x
     y de xevent.xbutton.y
  (etc.)
 Fin Bucle Eventos
```

Otra estructura útil es XAnyEvent, que incluye los cinco campos comunes a todas las estructuras de evento:

```
typedef struct
{
 int type;
 unsigned long serial;
 Bool send_event;
 Display *display;
 Window window;
} XAnyEvent;
```

El parámetro *type* es el tipo de evento, *serial* es el número de la última petición atendida, *send_event* es verdadero si procede de un SendEvent y falso en caso contrario, y *display* y *window* son el display y ventana de la que procede el evento.

La estructura XEvent es el paquete de información que el servidor envía al cliente

Mediante todos los datos vistos hasta ahora es posible averiguar el tipo de evento ocurrido y datos acerca del mismo, como se puede ver en el siguiente ejemplo:

```
XEvent evento;

while(1)
{
 CogerSiguienteEvento(evento);
 switch( evento.type )
 {
  case KeyPress:
       tecla = evento.xkey.xkeycode;
       break;
  case ButtonPress:
       etc etc etc...
 }
}
```

De este modo, ante cada evento se ejecutará el código correspondiente a lo que nosotros queremos que nuestra aplicación haga ante esa situación. Si se realizara un sencillo juego de naves, el control sería algo similar a lo siguiente:

```
 case KeyPress:
  tecla = evento.xkey.xkeycode;
  if( tecla == DERECHA ) x_nave++;
  else if (tecla = IZQUIERDA)
x_nave—;
  etc...
```

Ahora la pregunta principal debe centrarse en: ¿cómo se obtienen los eventos desde la cola de eventos hasta nuestra estructura XEvent?

RECOGIDA DE EVENTOS

Existen diferentes funciones para la recogida de eventos de la cola. Dependiendo del modo en que queramos que sean recogidos, disponemos de:

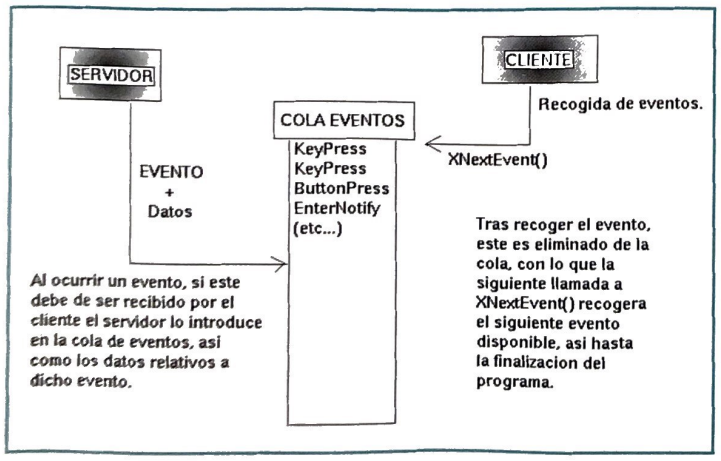
*XNextEvent(Display *display, XEvent *evento);*

Recoge el siguiente evento de la cola en modo bloqueante, es decir, recoge cualquier evento de cualquier ventana de nuestra aplicación y lo introduce en la estructura XEvent que se le pasa como parámetro. Es la más sencilla y más utilizada.

*XWindowEvent(Display *display, Window ventana, long Máscara, XEvent *evento);*
*XMaskEvent(Display *display, long máscara, XEvent *evento);*

Recogida de los eventos sólo de la ventana especificada o de la máscara de evento especificado (por si sólo quisieramos recoger eventos relacionados con *ButtonPressMask*, por ejemplo).

FIGURA 2. ENCOLADO Y RECOGIDA DE EVENTOS.



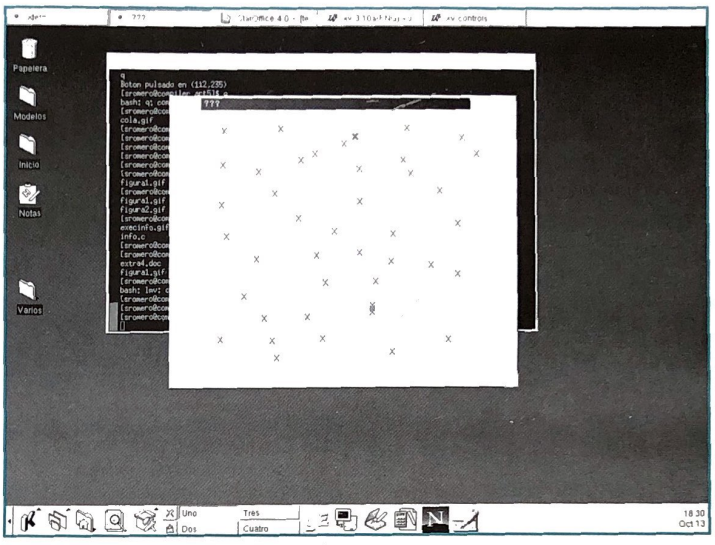


FIGURA 3. SALIDA DEL PROGRAMA TEST2.

*XCheckMaskEvent(Display *display, long máscara, XEvent *evento)*
*XCheckWindowEvent(Display *display, Window ventana, long máscara, XEvent *evento)*
*XCheckTypeEvent(Display *display, int tipo, XEvent *evento)*
*XCheckTypedWindowEvent(Display *display, Window ventana, int tipo, XEvent *evento)*

Conjunto de rutinas no bloqueantes, que requieren algunas condiciones concretas. Su utilidad se basa, por ejemplo, en recoger sólo los eventos de un tipo en concreto (*XCheckMaskEvent/ XCheckTypeEvent* y la máscara de eventos que se comentó en la anterior entrega o directamente el tipo de evento), sólo los eventos de una ventana en concreto (*XCheckWindowEvent()*), o los que cumplan ambas condiciones, ser de un tipo en concreto y pertenecer a la ventana especificada (*XCheckTyped WindowEvent()*). Como ejemplo, mediante esta última función sería posible recuperar únicamente (por si fueran los únicos que nos interesaran) los eventos de un tipo determinado y de una ventana en concreto (ej: sólo procesar el teclado en una ventana).

*XPeekEvent(Display *display, XEvent *evento);*

Resultado similar al de *XnextEvent*, pero no elimina el evento de la cola al recogerlo, simplemente lo lee e introduce sus datos en la estructura.
La diferencia entre un método bloqueante y uno que no lo es radica en que los primeros vacían el buffer de peticiones y esperan a que llegue un evento, mientras que los segundos cogen el evento si éste está presente, regresando sin esperar si no lo está.

XanyEvent incluye los cinco campos comunes a todas las estructuras de evento

A todas estas funciones se les pasa un parámetro XEvent que es donde éstas introducirán los datos del evento leídos desde la cola. Otras funciones existentes para la lectura son las siguientes (aunque no serán comentadas al no ser generalmente necesarias): *XIfEvent()*, *XCheckIfEvent()* y *XPeekIfEvent()*, que corresponden a eventos con condiciones definidas por el usuario (para dotar de más posibilidades al total de once funciones disponibles para la recuperación de eventos). Veamos un ejemplo de recogida de eventos (recordemos que la

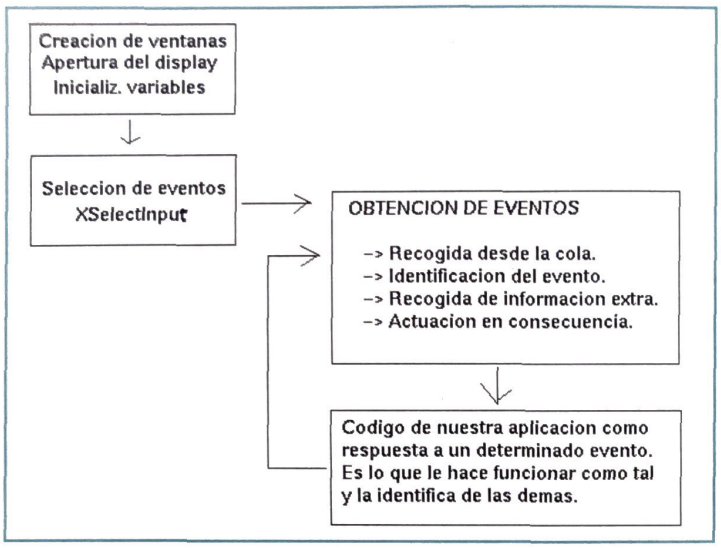


FIGURA 4. APLICACION ORIENTADA A EVENTOS.

compilación se realiza mediante *gcc test1.c -o test1.exe -lX11 -L/usr/X11R6/lib*):

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

void main()
{
 /* declaración de variables */
 Display *display;
 Window window;
 int pantalla;
 unsigned long white, black;
 XEvent evento;
 char tecla;

 /* inicializaciones */
 display = XOpenDisplay( NULL );
 pantalla = DefaultScreen( display );
 black=BlackPixel( display, pantalla );
 white=WhitePixel( display, pantalla );

 /* creamos la ventana */

window=XCreateSimpleWindow(display,

DefaultRootWindow(display),

0,0,400,300,5,white,black);
 /* seleccionamos los eventos deseados */
 XSelectInput(display, window, ButtonPressMask|KeyPressMask);
 XMapWindow( display, window );

 /* Bucle infinito de eventos */
 while(1)
 {
  /* Coger el evento siguiente en nuestra variable de eventos */
  /* Solo nos son enviados eventos que pedimos en XselectInput */
  XNextEvent(display, &evento);

  switch(evento.type)     /* ¿que evento ha ocurrido? */
  {
   case ButtonPress:   /* ¿boton del raton? */
     printf("Boton pulsado en (%i,%i)\n",
          evento.xbutton.x, evento.xbutton.y);
          break;

   case KeyPress:    /* ¿fue una tecla? */
     exit(0);      /* salir */
  }
 }
}
```

XWindow dispone de una única estructura para todos los mensajes (unión de las 29 existentes)

Ahora puede observarse cómo encaja todo lo explicado en los tres últimos meses: los eventos son enviados por el servidor, y nuestro programa cliente ya es capaz de recogerlos, identificarlos y actuar en consecuencia, como se muestra en la figura 2. Si se conocieran funciones gráficas, ahora sería posible hacer sencillos programas de dibujo que tracen *pixels* ante pulsaciones del mouse, hacer sencillas utilidades o programas de texto. Es posible crear para ello funciones que emulen un sistema de texto virtual en la ventana gráfica.

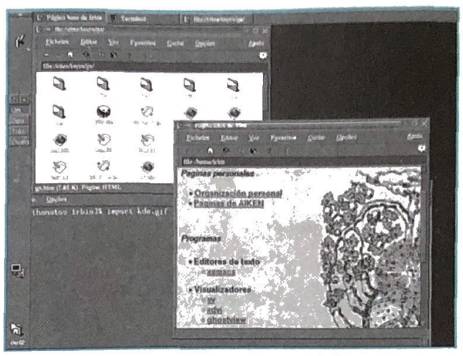


FIGURA 5. EL GESTOR DE VENTANAS KDE ES EN LA ACTUALIDAD EL MAS AVANZADO.

Las rutinas no bloqueantes regresan inmediatamente de la cola si no hay eventos disponibles

En el listado 1 podemos observar otro ejemplo de uso de eventos, esta vez con escritura en pantalla mediante *XSelectFont*, *XSetFont* y *XDrawString*. En este ejemplo se dibuja una X en la coordenada del mouse cada vez que un botón del mismo es pulsado, y se sale al pulsar cualquier tecla.

CAMPOS DE LAS DIFERENTES ESTRUCTURAS

Si necesita conocer los campos de las diferentes estructuras incluidas en XEvent, para averiguar cómo se sabe que *xkey* tiene un campo llamado *xkeycode* que contiene un código identificativo de la tecla pulsada puede utilizar *man* seguido de las diferentes estructuras disponibles (*XAnyEvent*, *XKeyEvent*, *XButtonEvent*, *XMotionEvent*, etc., consultables en *XEvent*). La salida del comando *man XkeyEvent* sería la siguiente:

```
typedef struct {
 int type;          /* KeyPress or KeyRelease */
 unsigned long serial;   /* # of last request processed */
 Bool send_event;       /* true if this came from a SendEvent */
 Display *display;      /* Display the event was read from */
 Window window;         /* event window it is reported relative to */
 Window root;           /* root window that the event occurred on */
 Window subwindow;      /* child window */
 Time time;             /* milliseconds */
 int x, y;              /* pointer x, y coordinates in event window */
 int x_root, y_root;    /* coordinates relative to root */
 unsigned int state;    /* key or button mask */
 unsigned int keycode;  /* detail */
 Bool same_screen;      /* same screen flag */
 } XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

En resumen

Gracias a lo visto hasta ahora pueden realizarse sencillos programas y ya se dispone de la base de programación en X-Window (y, en general, de la programación orientada a eventos en cualquier sistema, cuyo resumen puede observarse en la figura 4). En la próxima entrega ampliaremos nuestros conocimientos de programación en X-Window, ya que empezaremos a trabajar con gráficos.