Operating systems. Lecture 15

Michał Goliński

2019-01-22

Introduction

Recall

- Architecture of Windows
- PowerShell

Plan for today

- X Window System
  - The X protocol
  - Implementations
  - Future
- Sample programs
  - XCB – using the X protocol directly
  - Qt – Using a modern toolkit

X Window System

Introduction

Under Windows an application has to decide whether it is a command line application or a graphical application. The header of the executable file contains one of two flags: `IMAGE_SUBSYSTEM_WINDOWS_GUI` or `IMAGE_SUBSYSTEM_WINDOWS_CUI`. If a program marks itself as GUI, then it has no associated console and cannot, e.g., read from the standard input.

Displaying windows and its content is done by system calls.

Introduction cont.

Situation is different under Linux. When an application wants to display a window it needs to ask a specific program (the X server), to display a window on its account. The X server itself is not a part of the kernel, but novadays it is considered a core part of a Linux desktop.

Distributions that are meant to be installed on server may not have the X server installed at all.

History

The first version of the X server was created in 1984 at MIT. There was rapid progress and in 1986 already version X10R3 was created. MIT in order to make the software popular decided to make it available under what is now known as the MIT license.

X10R3 gained some popularity but it became apparent that some changes were still needed and in 1987 the eleventh version of the protocol together with an implementation were published. The protocol is still used today, known as X11.

History cont.

In 1988 MIT created the MIT X Consortium to steer the further development of X in a neutral way having both commercial and non-commercial aim in mind. X is one of the first large-scale open source projects.

In 1993 The X Consortium was formed, which lasted until 1996 and produced X11R6.3 in 1996. Stewardship of X was passed to The Open Group, and X11R6.4 was released in 1998 – this was a first version that was free to use only for noncommercial applications. This was later changed.

History cont.

At this point the development of the official X server mostly stalled. Most innovation happened in another implementation (used in all the Linux distributions at the time): XFree86. In 2004 XFree86 release a new version under a new license that was deemed too restrictive. This started the decline of XFree86.

History cont.

A group of developers started in 2004 the X.Org Foundation, got the x.org domain frrom the Open Group and forked XFree86 ander a new name: the X.Org Server. This attracted many developers in a more open, merit-based, development model. It is the most widely used implementation of X today.

The X protocol

The protocol has two sides: the server (draws windows) and client (the application that wishes to show windows). In principle it is not required that both run on the same machine. X protocol used to be network transparent, although some applications would have difficulty in running over a network.

Messages

There are four different types of messages:

- requests: client-originated, asks for information or requests actions.
- replies: server responds do requests
- events: server informs the client tat something happen (e.g., window was resized)
- error: server informs of an invalid request

Windows

All the drawing is done inside windows. A window may include other subwindows. Graphical elements like buttons menus etc. used to be subwindows. This gives us a tree that can be queried by:

```
xwininfo -tree -root
```

The most important are *top-level windows* that correspond to what the user perceives as different windows.

Window manager

The window manager is a program (distinct form the X server) that arranges windows on the screen. It allows the user to move and resize windows. It is also usually responsible for drawing the windows decorations (title bar etc.). There are two type of window managers:

- stacking: windows may obscure one another (the idea most people are accustomed to)
- tiling: divides the screen between all the active programs

Replacements

The X Window System is old and some of its features (e.g, network transparency) are not very helpful today, while their presence still has negative impact on performance.

Now several replacements of X are developed/used:

- macOS: Quartz
- Android: SurfaceFlinger

Future

Under Linux the most important competitor (and probably the future standard display server) is Wayloand. It uses a different protocol, shedding many layers that were added over the years and keeps/adds only those functionalities that are relevant today (e.g, more security). It should need less computations to achieve the same end result.

Sample programs

Xlib

Writing a program that speaks the X protocol directly can be problematic. Thankfully there are libraries that help with that. The Xlib used to be the standard way to interact with the X server. It keeps a queue of messages and most communication occurs in a serial way. Even though it might be easier to start programming with Xlib, in the long run some of the architectural choices made by the library can be problematic. Xlib is, e.g., difficult to use in a multithreaded environment, programmer usually needs to make sure only one thread handles all the X messages.

XCB

The X C bindings (xcb) is a new way to interact with the X protocol. It is much more fine grained. On one hand it requires more caution on behalf of the programmer, on the other hand it is much more flexible.

Xlib example

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

int main() {
    int width = 100;
    int height = 100;

    Display *display = XOpenDisplay(NULL);
    if(display == NULL) {
        printf("Cannot open display\n");
        exit(1);
    }
    int screen = DefaultScreen(display);
    Window window = XCreateSimpleWindow(display,
↪   RootWindow(display, screen), 10, 10, width, height, 1,
                                  BlackPixel(display, screen),
↪   WhitePixel(display, screen));

    XSizeHints *sizeHints = XAllocSizeHints();
    sizeHints->flags = PMinSize;
    sizeHints->min_width = 50;
    sizeHints->min_height = 50;
    XSetWMNormalHints(display, window, sizeHints);
```

```
    Atom delWindow = XInternAtom( display, "WM_DELETE_WINDOW", 0
↪   );
    XSetWMProtocols(display, window, &delWindow, 1);

    XSelectInput(display, window, ExposureMask | KeyPressMask |
↪   StructureNotifyMask);

    XMapWindow(display, window);

    while(1) {
        XEvent event;
        XNextEvent(display, &event);
        if(event.type == Expose) {
            XFillRectangle(display, window, DefaultGC(display,
↪   screen), 10, 10, width - 20, height - 20);
        } else if (event.type == ConfigureNotify) {
            XConfigureEvent xcevent = event.xconfigure;
            width = xcevent.width;
            height = xcevent.height;
        } else if(event.type == KeyPress) {
            break;
        } else if(event.type == ClientMessage) {
            break;
        }
    }

    XDestroyWindow(display, window);
    XCloseDisplay(display);

    return 0;
}
```

XCB example

```
#include <stdio.h>
#include <stdlib.h>

#include <xcb/xcb.h>
#include <xcb/xcb_icccm.h>

int main(void) {
    xcb_generic_event_t *e;
    int width = 100;
    int height = 100;
    xcb_rectangle_t      r = { 10, 10, width-20, height-20 };
```

```
    xcb_connection_t *c = xcb_connect(NULL, NULL);
    if (xcb_connection_has_error(c)) {
        printf("Cannot open display\n");
        exit(1);
    }
    xcb_screen_t *s =
↪   xcb_setup_roots_iterator(xcb_get_setup(c)).data;

    xcb_gcontext_t g = xcb_generate_id(c);
    xcb_window_t w = s->root;
    uint32_t mask = XCB_GC_FOREGROUND |
     ↪  XCB_GC_GRAPHICS_EXPOSURES;
    uint32_t values[2];
    values[0] = s->black_pixel;
    values[1] = 0;
    xcb_create_gc(c, g, w, mask, values);

    w = xcb_generate_id(c);
    mask = XCB_CW_BACK_PIXEL | XCB_CW_EVENT_MASK;
    values[0] = s->white_pixel;
    values[1] = XCB_EVENT_MASK_EXPOSURE |
↪   XCB_EVENT_MASK_KEY_PRESS | XCB_EVENT_MASK_STRUCTURE_NOTIFY;
    xcb_create_window(c, s->root_depth, w, s->root,
                      10, 10, width, height, 1,
                      XCB_WINDOW_CLASS_INPUT_OUTPUT,
↪   s->root_visual,
                      mask, values);

    xcb_size_hints_t hints;
    xcb_icccm_size_hints_set_min_size(&hints, 50, 50);
    xcb_icccm_size_hints_set_max_size(&hints, 5000, 5000);
    xcb_icccm_set_wm_size_hints(c, w, XCB_ATOM_WM_NORMAL_HINTS,
↪   &hints);

    xcb_map_window(c, w);

    xcb_flush(c);

    /* event loop */
    int done = 0;
    while (!done && (e = xcb_wait_for_event(c))) {
        printf("%d\n", e->response_type & ~0x80);
        switch (e->response_type & ~0x80) {
        case XCB_EXPOSE:    /* draw or redraw the window */
            xcb_poly_fill_rectangle(c, w, g,  1, &r);
            xcb_flush(c);
            break;
        case XCB_CONFIGURE_NOTIFY: ;
```

```
            xcb_configure_notify_event_t *nevent =
↪   (xcb_configure_notify_event_t *) e;
            width = nevent->width;
            height = nevent->height;
            r.width = width - 20;
            r.height = height - 20;
            break;
        case XCB_KEY_PRESS:  /* exit on key press */
            done = 1;
            break;
        }
        free(e);
    }
    /* close connection to server */
    xcb_disconnect(c);

    return 0;
}
```

Toolkits

Do not use the X protocol

Although the X protocol is the underlying mechanism to display windows under LInux, it should not generally be used by programmers. Using it properly is tedious and makes the program non-portable to Windows. End programmers should probably use **toolkits**, which help in building user interfaces. Linux has two popular toolkits: Qt and GTK, but these are by no means the only two.

Qt

Qt is C++ cross-platform framework/toolkit for creating GUI application. An application written with Qt can be run under Linux, Windows and MacOS (but will need to be re-compiled). Even though Qt is solely focused on C++, there are bindings available for other languages, most notably the official first-party bindings to Python (PySide).

Qt cont.

Qt gives tools to work with GUI, network, multimedia, databases, printing etc. This allows to create cross-platform programs with a single codebase easily. In my humble opinion the Qt toolkit is the most comfortable way to write GUI programs, not only in C++, but *in any language*.

Qt uses object-oriented programming paradigm.

A very simple applications

```
#!/usr/bin/env python

############################################################################
##
## Copyright (C) 2004-2005 Trolltech AS. All rights reserved.
##
## This file is part of the example classes of the Qt Toolkit.
##
## This file may be used under the terms of the GNU General
 ↪  Public
## License version 2.0 as published by the Free Software
 ↪  Foundation
## and appearing in the file LICENSE.GPL included in the
 ↪  packaging of
## this file.  Please review the following information to ensure
 ↪  GNU
## General Public Licensing requirements will be met:
## http://www.trolltech.com/products/qt/opensource.html
##
## If you are unsure which license is appropriate for your use,
 ↪  please
## review the following information:
## http://www.trolltech.com/products/qt/licensing.html or
 ↪  contact the
## sales department at sales@trolltech.com.
##
## This file is provided AS IS with NO WARRANTY OF ANY KIND,
 ↪  INCLUDING THE
## WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A
 ↪  PARTICULAR PURPOSE.
##
############################################################################

from PySide2 import QtCore, QtGui, QtWidgets


class DigitalClock(QtWidgets.QLCDNumber):
    def __init__(self, parent=None):
        super(DigitalClock, self).__init__(parent)

        self.setSegmentStyle(QtWidgets.QLCDNumber.Filled)

        timer = QtCore.QTimer(self)
        timer.timeout.connect(self.showTime)
        timer.start(1000)
```

```
        self.showTime()

        self.setWindowTitle("Digital Clock")
        self.resize(150, 60)

    def showTime(self):
        time = QtCore.QTime.currentTime()
        text = time.toString('hh:mm')
        if (time.second() % 2) == 0:
            text = text[:2] + ' ' + text[3:]

        self.display(text)


if __name__ == '__main__':

    import sys

    app = QtWidgets.QApplication(sys.argv)
    clock = DigitalClock()
    clock.show()
    sys.exit(app.exec_())
```