

Designing a Modern Window System

By George Peter Staplin

Feb 2008

1. The goals of a modern window system.
2. Problems and solutions
3. Security
4. Implementation

1. The goals of a modern window system

A modern window system should provide:

1. a standard extensible widget toolkit

Without a standard toolkit users are often confused. While the X window system has provided for many experimental interfaces, it has also lead to many evolutionary architectural splits in the world of toolkits. With a standard button widget for example you can provide more features, and waste less time with support. The widget toolkit should be extensible from the client. This means that many widgets are client-side. The reason this was chosen was to eliminate the need to update the server.

2. fast transfers of graphics for video

This usually requires shared memory for medium to large videos. The X window system has provided a variety of interfaces (Xv, Xshm, etc.) that have made it difficult to program for the system. There are many ways of transferring the graphics to the display, and ideally a program should support them all, with support for falling back to another extension when one is unavailable, with the last being the core XPutImage().

The best way of solving the problem with the current architectural restrictions is shared memory. Memory mapped files can do this well, and provide for a shared and easier to debug representation of the screen and window contents.

3. a simple API

The API should not hinder or make it overly difficult to treat the window contents as a frame-buffer.

All of the basic raster rendering algorithms should be implementable with a client window.

4. automatic double-buffering

Double-buffering is a requirement these days. It's unacceptable to have flicker. The programmer should not have to think about the implementation of the core window system. Simply put: manual double-buffering is an indication of a missing abstraction.

5. optional network display

Most displays are not networked, however it's often useful to have remote display. The remote framebuffer protocols, and VNC are quite useful for this task. Some have proven to be faster than X11 over fast links. The window system should allow for remote control and sharing a desktop with someone over the network.

6. X11 compatibility

X11 is a popular system. This is why we need the ability to at least display and interact with X11 windows. A modified Xvfb-like program can be used to provide a new window system application. This would provide an X server, similar in function to Xnest.

7. Drivers and modules as separate processes

Drivers should be easily upgradeable, with a test mode. The core window system shouldn't care how the driver is implemented. A video driver can use shared memory to copy the contents of the window using the most efficient means possible to transfer the contents to the video card memory. The pipe() abstraction can be used as a channel between video drivers. A mouse driver should be as simple as possible, with a similar architecture. In fact it may be possible to insert a filter between a mouse driver for filtering, and likewise with keyboards.

2. Problems and solutions

The usage of shared memory for every window in the tree, results in unacceptable performance with some systems, such as a Pentium® 4-based desktop. A simple solution to this problem is to provide each top-level window a shared memory buffer, with the children of this shared memory buffer being allocated with malloc() or a similar allocator. So, essentially the compositing for a client occurs in the client process. This has proven to scale acceptably, but perhaps there is some opportunity for acceleration of the compositing of client-side windows.

A means of sending requests to a separate process must be provided. For the experimental implementation I have chosen the use of the socket layer provided by unix-like systems. The basic core window system server communicates with the applications using sockets.

An event queue is also needed, or some means of notifying an application's thread-like entities of buttons being pressed, and things of that nature. This could be implemented as part of the protocol used with sockets, or a FIFO (First in First Out – shared file) can be used. At this time it's not known which is currently preferable, but a FIFO is used in the experimental implementation.

3. Security

One major problem with the current window systems is security. X11 has suffered from local-host exploits that have resulted in system administrators disabling X11 in many cases on untrusted servers.

Security presents several problems with this design, but they are all curable.

1. The design of the memory mapped file facilities allows for the client to truncate the file that provides the backing memory for the window contents. If not handled properly the server can fault during an update of the window contents.

The solution to this problem is to use read() on the server-side. This makes it so that even if a client truncates its window file, it's unable cause a fault in the server. This can also be used to provide a form of double-buffering.

2. A form of access control should be implemented to prevent foreign clients from causing a Denial of Service Attack.

One solution to this is simple. We don't allow non-local-host clients at the moment. A better design should be provided, with perhaps a standard dialog provided to ask the user if they want to allow a client.

3. It should be impossible for a client to access the contents of another client.

This will prevent the recording of critical data, such as passwords. All transferring of data must be explicit. The design of the X window system allows any application free reign. A foreign application can destroy other windows, read input, grab input, etc. This is not a trustworthy model.