

## computers are bad

You are receiving this facsimile because you signed up for fax delivery of this newsletter. To stop delivery, contact Computers Are Bad by email or fax.

<https://computer.rip> - [me@computer.rip](mailto:me@computer.rip) - fax: +1 (505) 926-5492

---

**2021-09-08 W X Y Z**

Let's return, for a while, to the green-ish-sometimes pastures of GUI systems. To get to one of my favorite parts of the story, delivery of GUIs to terminals over the network, a natural starting point is to discuss an arcane, ancient GUI system that came out of academia and became rather influential.

I am referring of course to the successor of W on V: X.

Volumes could be written about X, and indeed they have. So I'm not intending to present anything like a thorough history of X, but I do want to address some interesting aspects of X's design and some interesting applications. Before we talk about X history, though, it might be useful to understand the broader landscape of GUI systems on UNIX-like operating systems, though, because so far we've talked about the DOS/VMS sort of family instead and there are some significant differences.

Operating systems can be broadly categorized as single-user and multi-user. Today, single-user operating systems are mostly associated only with embedded and other very lightweight devices [1]. Back in the 1980s, though, this divide was much more important. Multi-user operating systems were associated with "big iron," machines that were so expensive that you would need to share them for budget reasons. Most personal computers were not expected to handle multiple users, over the network or otherwise, and so the operating systems had no features to support this (no concept of user process contexts, permissions, etc).

Of course, you can likely imagine that the latter situation, single-user operating systems, made the development of GUI software appreciably easier. It wasn't even so much about the number of users, but rather about where they were. Single-user operating systems usually only supported someone working right at the console, and so applications could write directly to the graphics hardware. A windowing system, at the most basic, only really needed to concern itself with getting applications to write to the correct section of the frame buffer.

On a multi-user system, on the other hand, there were multiple terminals connected by some method or other that almost certainly did not allow for direct memory access to the graphics hardware. Further, the system needed to manage what applications belonged on which graphics devices, as well as the basic issue of windowing. This required a more complicated design. In particular, server-client systems were extremely *in* at the time because they had the same general shape as the computer-terminal architecture of the system. This made them easier to reason about and implement.

So, graphics systems written for multi-user systems were often, but not always,

server-client. X is no different: the basic architecture of X is that of a server (running within the user context generally) that has access to a graphics device and input devices (that it knows how to use), and one or more clients that want to display graphics. The clients, which are the applications the user is using, tell the server what they want to display. In turn, the server tells the clients what input they have received. The clients never interact directly with the display or input hardware, which allows X to manage multiple access and to provide abstraction.

While X was neither the first graphics system for multi-user operating systems, nor the first server-client graphics system, it rapidly spread between academic institutions and so became a de facto standard fairly quickly [2]. X's dominance lasts nearly to this day, Wayland has only recently begun to exceed it in popularity. Wayland is based on essentially the same architecture.

X has a number of interesting properties and eccentricities. One of the first interesting things many people discover about X is that its client-server nature actually means something in practice: it is possible for clients to connect to an X server running on a different machine via network sockets. Combined with SSH's ability to tunnel arbitrary network traffic, this means that nearly all Linux systems have a basic "remote application" (and even full remote desktop) capability built in. Everyone is very excited when they first learn this fact, until they give it a try and discover that the X protocol is so hopelessly inefficient and modern applications so complex that X is utterly unusable for most applications over most internet connections.

This gets at the first major criticism of X: the protocol that clients use to describe their output to X is very low level. Besides making the X protocol fairly inefficient for conveying typical "buttons and forms" graphics, X's lack of higher-level constructs is a major contributor to the inconsistent look-and-feel and interface of typical Linux systems. A lot of basic functionality that *feels* cross-cutting, like copy-and-paste, is basically considered a client-side problem (and you can probably see how this leads to the situation where Linux systems commonly have two or three distinct copy and paste buffers).

But, to be fair, X never aimed to be a desktop environment, and that type of functionality was always assumed to occur at a higher level.

One of the earliest prominent pseudo-standards built on top of X was Motif, which was used as a basis for the pseudo-standard Common Desktop Environment presented by many popular UNIX machines. Motif was designed in the late '80s and shows it, but it was popular and both laid groundwork and matched the existing designs (Apple Lisa etc) to an extent that a modern user wouldn't have much trouble using a Motif system.

Motif could have remained a common standard into the modern era, and we can imagine a scenario where Linux had a more consistent look-and-feel because Motif rose to the same level of universality as X. But it didn't. There are a few reasons, but probably the biggest is that Motif was proprietary and not released as open source until well after it had fallen out of popularity. No one outside of the big UNIX vendors wanted to pay the license fees.

There were several other popular GUI toolkits built on top of X in the early days, but I won't spend time discussing them for the same reason I don't care to talk about Gnome and KDE in this post. But rest assured that there is a complex and often frustrating history of different projects coming and going, with few efforts standing the test of time.

Instead of going on about that, I want to dig a bit more into some of the less discussed implications of X's client-server nature. The client and server could exist on different machines, and because of the hardware-independence of the X protocol could also be running different operating systems, architectures, etc. This gave X a rather interesting property: you could use one computer to "look at" X software running on another computer almost regardless of what the two computers were running.

In effect, this provided one of the first forms of remote application delivery. Much like textual terminals had allowed the user to be physically removed from the computer, X allowed the machine that rendered the application and collected input to be physically removed from the actual computational resources running the software. In effect, it created a new kind of terminal: a "dumb" machine that did nothing but run an X server, with all applications running on another machine.

The terminology around this kind of thing can be confusing and is not well agreed upon, but most of the time the term "thin terminal" refers to exactly this: a machine that handles the basic mechanics of graphical output and user input but does not run any application software.

Because of the relatively high complexity of handling graphics outputs, thin terminals tend to be substantially similar to proper "computers," but have very limited local storage (usually only enough for configuration) and local processing and memory capacity that are just enough to handle the display task. They're like really low-end computers, basically, that just run the display server part of the system.

In a way that's not really very interesting, as it's conceptually very similar to the block terminals used by IBM and other mainframes. In practice, though, this GUI foray into terminals took on some very odd forms over the early history of personal computers. The terminal was decidedly obsolete, but was also the hot new thing.

Take, for example, DESQview. DESQview was a text-mode GUI for DOS that I believe I have mentioned before. After DESQview, the same developer, Quarterdeck, released DESQview/X. DESQview/X was just one of several X servers that ran on DOS. X was in many ways a poor fit for DOS, given DOS's single task nature and X's close association with larger machines running more capable operating systems, but it was really motivated by cost-savings. DOS was cheap, and running an X server on DOS allowed you to both more easily port applications written for big expensive computers, and to use a DOS machine as an X server for an application running on another machine. The cheap DOS PC became effectively a hybrid thin terminal that could both run DOS software and "connect to" software running on a more expensive system.

One way to take advantage of this functionality was reduced-cost workstations. For example, at one time years ago the middle school I attended briefly had a computer lab which consisted of workstations (passed down from another better funded middle school) with their disks removed. The machines booted via PXE into a minimal Linux environment. The user was presented with a specialized display manager that connected to a central server over the network to start a desktop environment.

The goal of this scheme was reduced cost. In practice, the system was dog slow [3] and the unfamiliarity of KDE, StarOffice, and the other common applications on the SuSE server was a major barrier to adoption. Like most K-12 schools at the time the middle school was already firmly in the grasp of Apple anyway.

Another interesting aspect of X is the way it relates to the user model. I will constrain myself here to modern Linux systems, because this situation has varied over

time. What user does X run as?

On a typical Linux distribution (that still uses X), the init system starts a desktop manager as root and then launches an X server to use, still with root privileges. The terminology for desktop *things* can get confusing, but a desktop *manager* is responsible for handling logins and setting up graphical user sessions. It's now common for the display manager to actually run as its own user to contain its capabilities (by switching users after start), so it may use `setuid` in order to start the X server with root capabilities.

Once a user authenticates to the display manager, a common display manager behavior is to launch the user's desktop environment as the user and then hand it the information necessary to connect to the existing X instance. X runs as root the whole time.

It is completely possible to run X as a non-privileged user. The problem is that X handles all of the hardware abstraction, so it needs direct write access to hardware. For numerous reasons this is typically constrained to root.

You can imagine that the security concerns related to running X as root are significant. There is work to change this situation, such as the kernel mode setting feature, but of course there is a substantial problem of inertia: since X has always run with root privileges, a great many things assume that X has them, so there are a lot of little problems that need solving and this usage is not yet well supported.

This puts X in a pretty weird situation. It is a *system* service because it needs to interact directly with scarce hardware, but it's also a *user* application because it is tied to one user session (ultimately by means of password authentication, the so-called X cookie). This is an inherent tension that arises from the nature of graphics cards as devices that expect very low-level interaction. Unfortunately, continuously increasing performance requirements for graphical software make it very difficult to change this situation... as is, many applications use something like mesa to actually bypass the X server and talk directly to the graphics hardware instead.

I am avoiding talking about the landscape of remote applications on Windows, because that's a topic that deserves its own post. And of course X is a fertile field for technology stores, and I haven't even gotten into the odd politics of Linux's multiple historic X implementations.

[1] Windows looks and feels like a single-user operating system to the extent that I sometimes have to point out to people that NT windows releases are fully multi-user. In fact, in some ways Windows NT is "more multi-user" than Linux, since it was developed later on and the multi-user concept is more thoroughly integrated into the product. Eventually I will probably write about some impacts of these differences, but the most obvious is screen locking: on Linux, the screen is "locked" by covering it with a window that won't go away. On Windows, the screen is "locked" by detaching the user session from the local console. This is less prone to bypasses, which perennially appear in Linux screensaver implementations.

[2] The history of X, multi-user operating systems from which UNIX inherited, and network computing systems in general is closely tied to major projects at a small number of prominent American universities. These universities tended to be very ambitious in their efforts to provide a "unified environment" which lead to the development of what we might now call a "network environment," in the sense of shared resources across an institution. The fact that this whole concept came out of

prominent university CS departments helps to explain why most of the major components are open source but hilariously complex and notoriously hard to support, which is why everyone today just pays for Microsoft Active Directory, including those universities.

[3] Given the project's small budget, I think the server was just under-spec'd to handle 30 sessions of Firefox at once. The irony is, of course, that as computers have sped up web browsers have as well, to the extent that running tens of user sessions of Firefox remains a formidable task today.

Note: several corrections made, mostly minor, thanks to HN users smcameron, yrro, segfaultbuser. One was a larger one: I find the startup process around X to be confusing (you hopefully see why), and indeed I described it wrong. The display manager starts first and is responsible for starting an X server to use, not the other way around (of course, when you use xinit to start X after login, it goes the way I originally said... I didn't even get into that).