1

Introduction

The X Window System was originally developed in the early 1980's, and encompassed from the beginning many of the windowing capabilities that we now take for granted. While in a number of ways X was (and still is) command-line oriented, the capability of moving away from the command line was inherent from the very beginning in the architecture of the system. The advent of the desktop graphical user interface (GUI) didn't require a major redesign of the X Window System.

Figure 1-1 illustrates what an X desktop might have looked like in the early days.

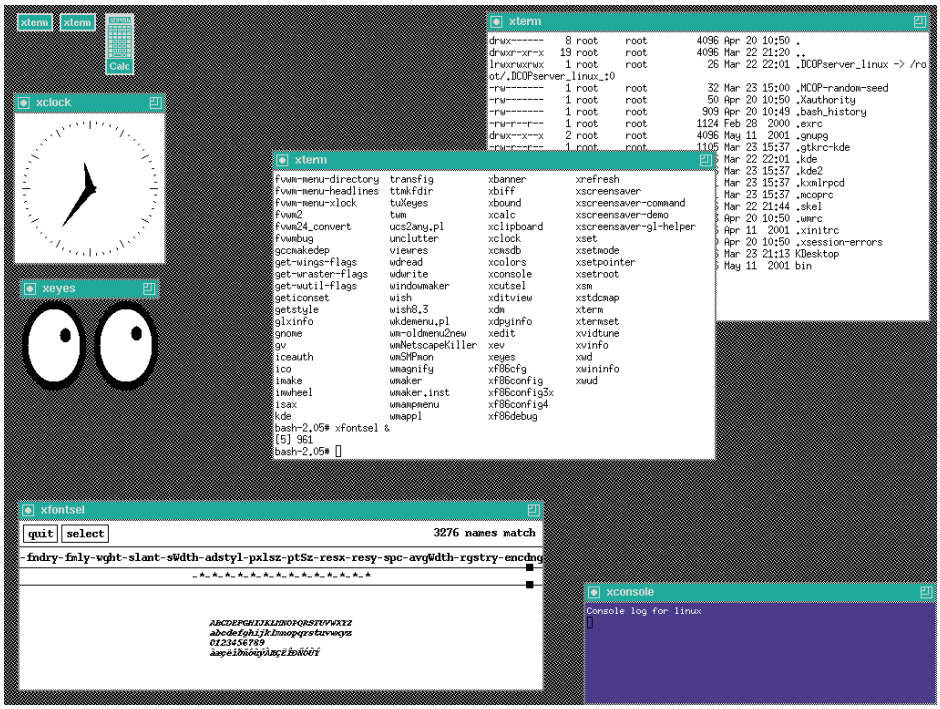

Figure 1-1: X desktop in the early days using twm

But times have changed. Shown in Figure 1-2 is what a modern X desktop can now look like. This example uses the KDE Desktop Environment described later in Chapter 9, *Using KDE*.

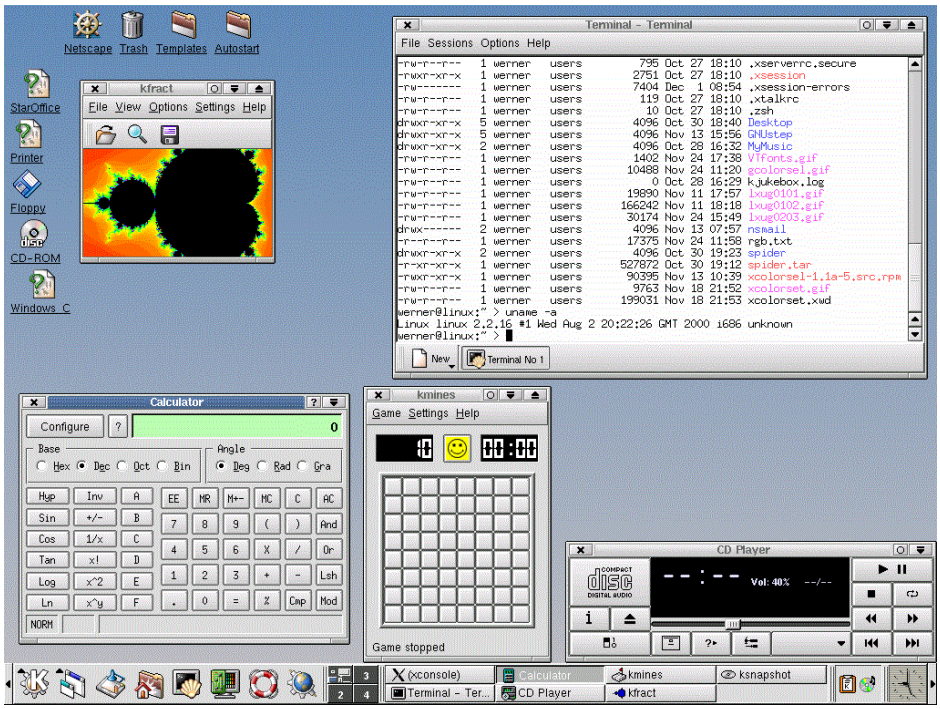

Figure 1-2: Modern X desktop using KDE

Quite different in appearance--the more modern example has a fancier desktop and is visually more appealing--but technically there's little difference between these examples. The X server still communicates with the X client via the X protocol over a network, and a window manager is still being used to manage the client application windows. The basics haven't changed, just the frills.

Part I of this book describes the underlying features of X that make it such a versatile and enduring system; Part II takes a look at some of the modern window managers and the two major desktop environments, GNOME and KDE; and Part III puts the theory, which sometimes needs configuration help and effort, into practice. But before we move on, the rest of this chapter provides an overview of X, briefly describing its client/server architecture and the major features of the X server and of X client applications.

The X Window System, called X for short (*not* X Windows), is a network-based graphical window system that was developed at MIT in 1984. The project leader of the main development was Robert Scheifler. Why X and not some other letter? This is because the origins of X owe much debt to the "W" Windowing package developed by Paul Asente at Stanford University. The development of X was part of Project Athena, a cooperative effort between MIT, IBM, and Digital Equipment Corporation to develop a network of heterogeneous systems that could be used for teaching purposes. The current

version, X11, was first released in 1987 and is now up to X11 release 6, known more commonly as X11R6.

In 1993, MIT turned control of X over to the X Consortium to continue development. In 1997, the X Consortium transferred control of X over to another consortium, the Open Software Foundation (OSF), which in turn became The Open Group. Ownership and development of X remains with The Open Group today.

The XFree86 Project

The standard version of X on Linux is XFree86.* Development of XFree86 started in the early 1990's as an effort to port X to the PC architecture. Like Linux itself, XFree86 has now been ported to many other systems. Like X itself, XFree86 has always been distributed with an open-source license. The current version of XFree86, version 4.x.x, is based on The Open Group's X11R6.4. Note, though, that some Linux distributions still come with the last major version, XFree86 v3.3.6. If you want to upgrade, you can download the latest version from the XFree86 website at *www.xfree86.org*.

The XFree86 Project, Inc. was formally started in 1992 by a small group of developers intending to provide an X Window System for Intel-based PCs. Today's XFree86 distribution also supports many other platforms, such as DEC's (Compaq's) Alpha, Sun's SPARC, and Motorola's PowerPC CPU-based systems.

See Chapter 10, *Configuring XFree86*, for details on configuring the XFree86 server.

With the exception of Chapter 10, we generally refer in this book simply to X or the X Window System, rather than to XFree86. This is because most of the discussion applies to any version of X that is based on X11R6. Our assumption is that you are running XFree86 on your Linux system; however, much of the information in the book still applies even if you are running a different version of X.

X Architecture Overview

The X Window System is based on the *client/server* model. The system is divided into two parts:

The X display server
: Provides display capabilities, manages the display hardware, and manages user input.

X client applications
: Application programs that perform specific tasks.

Client/server technology as used by X may seem backwards at first glance. Generally, people think of servers as remote machines like file servers, news servers, or mail servers. For X, however, the server runs on the local machine, providing its services to the

---

* There are also some commercial versions available, such as Metro Link Inc.'s Metro-X or Xi Graphic Inc.'s Accelerated-X.

display, based on requests from client programs that may be running remotely or locally and on input from the user.

You might choose to run a client on a remote machine for any number of reasons. Generally, however, the remote machine offers some feature unavailable on your local machine: a bigger or faster processor, a different architecture better suited to a particular task, application software not available locally, or maybe file server capabilities. X allows you to take advantage of these remote features and to see the results on your local display.

One of the most significant features of X is that it was specifically designed to work across a network. The client and the server communicate via the *X Protocol*, which can run locally or over a TCP/IP or DECnet (for historical reasons) network in a way that is transparent to the user. Your local Linux system will run an X display server and some local clients; other clients may be run across the network. It's all the same to you sitting at your display.

In a sense, the server acts as intermediary between client application programs and the local display hardware and input devices (generally a keyboard and mouse or other pointing device). When you enter input using the keyboard or mouse, the server intercepts the input as a sequence of *events* and notifies the relevant client application. Likewise, clients make requests that are communicated to the hardware display by the server. For example, a client may request that a window be moved or that text be displayed in the window.

X was designed to make the software independent of the hardware. Special graphics terminals, known as X terminals, were developed to run X. Nowadays, depending on the type of system you're running Linux on, your X display is likely to be a standard PC or Macintosh monitor. As a windowing system, X allows you to work with multiple programs, each running in a separate window. For example, the display that we saw in Figure 1-1 shows an X session with six client windows open.

As we'll see throughout the book, the architecture of the X Window System results in a highly portable, configurable, and flexible windowing system.

The X Server

The X server is the program that manages the hardware interface to the display. The server accepts user input from devices such as the keyboard and the pointer and sends that input to the relevant client applications; the server also keeps track of client output and updates the display to reflect that output. Each physical display (which may be multiple screens) has only one server program running.

User input and several other types of information pass from the server to a client in the form of *events*. An event is a packet of information that tells the client something it needs to act on, such as keyboard input. Actions such as moving the pointer or pressing a key cause *input* events to occur. The server intercepts each event and passes it on to the client.

When a client program receives a meaningful event, it processes the event and then responds with a *request* to the server for some sort of action affecting the display. For example, the client may request that a window be resized to particular dimensions. The server responds to the client request updating the appropriate window on the display.

However, it's important to know that the window manager interprets this data. The window manager is responsible for the look and feel of the display and decides how the client window should be displayed.

The X Protocol, Xlib, and Toolkits

At the lowest level, the server and the client programs communicate using the X Protocol. In general, however, only programmers writing X server software need to use the X Protocol directly. For writing X clients, programmers use *libraries*, which are archives of commonly used functions.* As an X user, you don't need to be concerned about libraries and toolkits; however, you'll see the terms and should know what they mean.

Xlib is the lowest-level library for writing X client applications. Xlib defines an interface to the X Protocol, with a one-to-one correspondence between X protocol requests and Xlib library functions. Writing a program in Xlib is time-consuming and laborious; rather than having to go through that process, programmers use *toolkits* written on top of Xlib. A toolkit is a set of programming libraries and *widgets* for programming in X. A widget is best described as a graphical component, such as a menu, command button, dialog box, or scrollbar. Programmers use toolkits not only because it's easier than writing from scratch, but also so their applications have a uniform look and feel with other applications.

Popular toolkits for X11 include the original Athena toolkit developed by the X Consortium, Motif developed by the Open Software Foundation, OLIT developed by AT&T, and XView developed by Sun. Both OLIT and XView are implementations of the OPEN LOOK Graphical User Interface (GUI) specification. Some more modern toolkits include Qt from Trolltech, Inc. (the toolkit underlying KDE) and GTK (the GIMP Toolkit, underlying GNOME).

Although it defines a certain look and feel, a toolkit does not provide an environment. The idea is for vendors who license the toolkit to design their own environments on top of it. Examples of such environments are Sun OpenWindows (based on Xview) and products such as SCO Open Desktop and HP Vue (both based on Motif).

Many of the popular X window managers (client applications that take care of managing a user's windows), are strongly associated with a toolkit and have a design that's consistent with that toolkit. We'll talk a lot more about window managers later in this chapter, in Chapter 2, *Getting Started Using X*, and with definitive examples in Part II, *Window Managers and Desktop Environments*.

Standards for the interaction between X and the window managers and other clients are defined in the *Inter-Client Communication Conventions Manual* (ICCCM) This document defines basic policy intentionally omitted from X itself, such as the rules for transferring data between applications, for transferring keyboard focus, for installing colormaps, and so on. If window managers and other applications follow the conventions outlined in the ICCCM, they should be able to coexist and work together on the same

---

* It's beyond the scope of this book to describe programming for X, whether for the server itself or for client applications. If you plan on doing X programming, you might want to see the O'Reilly X Window System series for X-specific programming or *Linux in a Nutshell* for general Linux programming information.

server--even if they have been written using different toolkits. Not only do popular window managers such as FVWM and Enlightenment fulfill this standard, but so do desktop environments such as GNOME and KDE. This makes it possible, for example, to have KDE applications run inside GNOME or have Enlightenment be the preferred window manager for GNOME.

Session Management

Beginning with X11R6, X has provided support for *session management*. A session consists of the group of applications and their present configurations that the user is running. True session management enables the user to log out and log back in to find the same set of applications running in the same states, A *session manager* is a program analogous to the window manager that communicates with applications to keep track of how to restart them in their current states It also provides a user interface for logging out that allows the user to interact with each program to save work.

The session manager communicates with the applications using the *Session Management* (SM) protocol, which in turn is based on another standard, the *InterClient Exchange* (ICE) protocol. The session manager and the applications communicate with each other directly using ICE, not through the X server; this allows non-X programs to be part of a session and to be managed by the session manager. This means that the session manager would properly restart a non-X program.

Even without an actual session manager or a window manager that does session management, you can control the initial state of each login session by creating an appropriate startup file. The name of this file depends on how you log in. We'll clarify this in the next chapter but for the moment we'll take a look at an *.xsession* file in your home directory. This file lets you specify certain clients to be started automatically when you log in to X. For example, you might have an *.xsession* file that looks like this:

Example 1-1: sample .xsession *file*

```
#! /bin/bash

. ~/.bashrc
exec 2>&1 > $HOME/.xsession-errors
xrdb -load $HOME/.Xresources
xsetroot -grey &
xhost +coke
xclock -geometry +10+890 &
xbiff -geometry +1195+15 &
xload -geometry 150x60+200+927 &
xterm -geometry 80x50+1962+55 -n xterm@ruby &
xterm -geometry 80x50+20+25 -n xterm@ruby &
xterm -geometry 80x50+740+125 -n xterm@ruby &
fvwm2
```

We'll discuss the details of the *.xsession* file in Chapter 2; what is interesting now is to see that this file runs certain applications during login, placing their windows at particular screen locations (specified by the *-geometry* option). This is not true session management, but it does save you from having to start each application manually every time you log in.

Note, though, that this rudimentary session management doesn't provide for saving the state of the last session. During each login, you get the set of windows specified in the *.xsession* file, but no applications will be running in them. With real session management,

any applications that were open during the previous logout will be open when a new session is started.

X Clients

A *client* is an application program that runs under X. The standard release of X includes many client programs that perform a wide variety of tasks, and most Linux distributions include additional clients. X allows you to run multiple clients simultaneously, with each client displayed in a separate window. For example, you can be editing a text file in one window, compiling a program source file in a second window, reading your mail in a third, all the while displaying the system load average in a fourth window. And of course you might also want to have a clock application handy on your screen and a web browser running.

While X clients generally display their results and take input from a single display server, they may each be running on a different computer on the network. It is important to note that the same programs may not always look and act the same. There are several reasons for this:

- X has no standard user interface. The window manager being used has a strong influence on the client's look and feel.
- X clients can be customized by the user differently on each server, and the display hardware on each server may have different specifications and capabilities.

In addition to communicating with the server, a client sometimes needs to communicate with other clients. The most extreme example is the window manager. Though the window manager is itself a client that communicates and interacts with the X server, it also communicates with the clients running under it. For example, the window manager needs to get information from the client such as where to place the client's icon. Interclient communication is facilitated by the use of *properties*; a property is a piece of information associated with a window or a font and stored in a file accessible by the X server. Properties are used by clients to store information that other clients might need to know, such as the name of the application associated with a particular window. Storing properties in the server makes the information accessible to all clients.

Most clients come with a default set of properties. These defaults can be overridden on a system-wide basis, or individually by each user (see Chapter 12, *Setting Resources*). A typical use of properties in interclient communication involves the way a client tells the window manager the name of the application associated with its window. By default, the application name corresponds to the name of the client, but many clients allow you to specify an alternative name when you run the program. A window manager that provides a titlebar needs to know the application name to display in that area of the window. The client's application name is stored in the server in a property called `WM_NAME` and is retrieved from there by the window manager.

Managing Client Windows

The operations performed within a window can vary greatly, depending on the type of program running. Some windows wait for input from the user, functioning as terminals and allowing you to write and test programs, control a database, write a book, etc. Other

windows simply display information, such as the time of day or a picture of the characters in a particular font. X also lets you perform operations across windows; one of the most useful features of X is that you can cut and paste text between two windows.

One client type that you may use frequently is a *terminal emulator*, a window that functions as a standard terminal. The standard terminal emulator included with X is called *xterm*. Figure 1-1 contains four *xterm* windows. Two of these *xterm* windows are *iconified*, symbolizing that they represent a window in an inactive state. In an *xterm* window, you can do anything you might do in a regular terminal: enter commands, run editing sessions, compile programs, etc.

In addition to the four *xterm*s, the display in Figure 1-1 also includes five other application windows: a window that displays console messages (called *xconsole*), a clock (*xclock*), a pair of eyes that follow the pointer (*xeyes*), a font selector (*xfontsel*), and an iconified calculator (*xcalc*). X provides many such small utility programs intended to make your work easier, and some such as *xeyes* that simply make it more enjoyable.

The shaded area that fills the entire screen is called the *root* (or *background*) *window*. X client application windows are displayed on top of the root window. X treats windows as a hierarchy, similar to a family tree. The root window is the root or origin within this hierarchy and is considered to be the *parent* of application windows displayed on it. Conversely, the application windows are called *children* of the root window. In Figure 1-1, the *xterm*, *xconsole, xclock*, *xcalc*, *xfontsel*, and *xeyes* windows are all children of the root window.

The window hierarchy is actually much more complicated than this "two generation" model suggests. Various parts of the application windows are themselves windows. For example, many applications provide menus to assist the user. Technically speaking, these menus are separate windows. Understanding the complexity of the window hierarchy (and the composite parts of an application) will become important in Chapter 12, when we discuss how to tailor an application to suit your needs.

As we mentioned earlier, a program called the *window manager* takes care of managing client windows. The window manager controls the geometry and aesthetics of your X display, allowing you to perform such operations as changing the size and position of windows on the display. You can reshuffle windows in a window stack, make windows larger or smaller, move them to other locations on the screen, etc. The window manager provides part of the user interface--to a large extent it controls the "look and feel" of the X Window System.

Notice in Figure 1-1 that two of the *xterm* windows overlap each other. If you use many windows, you'll find that they often overlap, much like sheets of paper on your desk or a stack of cards. This overlapping does not interfere with the process running in each window, but you need a way to move and modify the windows on your display. For example, if you want to work in a window that is partially covered by another window, you need to be able to raise the obscured window to the top of the window stack.

Historically, the standard window manager was TWM, and even now on Linux systems it is generally the default window manager if no other window manager is available. Later, Linux came with FVWM as the standard window manager. The current version of FVWM is FVWM2; FVWM itself is no longer being actively developed, although it is

still available from the FVWM website at *www.fvwm.org*. For the remainder of Part I, we generally assume the use of FVWM2 for the purpose of our discussion.

Using the Pointer

All X displays require you to have some sort of *pointer*, generally a mouse with two or three buttons, with which you communicate information to the system. Although X assumes a three-button mouse by default, if you have a mouse with two buttons, you can set up your X environment to treat a simultaneous click of both buttons as though it were the third button. If you are running X on a Macintosh PowerPC, which uses a one-button mouse, you can use modifier keys on the keyboard to emulate the missing buttons. You can also use a mouse with a scroll-wheel or another device such as a trackball or a touchpad. See Chapter 10 for information on setting up the pointer for use with XFree86.

As you slide the pointer around on your desktop, a cursor symbol on the display follows the pointer's movement. For our purposes, we will refer to both the pointing device (e.g., a mouse) and the symbol that represents its location on the screen as pointers. Depending on where the pointer is on the screen, it is represented by one of a number of cursor symbols. If the pointer is positioned on the root window, it is generally represented by an X-shaped cursor. If the pointer is in an *xterm* window, it looks like an uppercase I and is commonly called an *I-beam cursor*.*

While you'll use the keyboard whenever you need to enter text, whether you're entering commands in an *xterm* or typing in an editor, you'll find that you use the pointer at least as often. Some of the things you'll use it for are:

- Selecting the active window; i.e., the window you are going to work in next.
- Managing windows: moving, resizing, raising, or lowering them.
- Iconifying or deiconifying windows.
- Accessing menu options.

We'll talk about each of these items in detail in Chapter 2.

Customizing Clients

Most X clients are designed to be customized by the user. A multitude of command-line options can be used to affect the appearance and operation of a single client process. Some of the more useful command-line options are introduced in Chapter 2.

X also provides a somewhat more convenient way to customize the appearance and operation of client programs. Rather than specifying all characteristics with command-line options, default values for most options can be stored in a file (generally called *.Xresources* or *.Xdefaults*) in your home directory.* Each default value is set using a

---

* Even though the actual image on the screen is called a cursor, throughout this book we'll refer to "moving the pointer" to avoid confusion with the standard text cursor that can appear in an *xterm* window.

* There are also system-wide defaults provided that apply if you don't have your own file set up.

variable called a *resource*; you can change the behavior or appearance of a program by changing the value associated with a resource variable.

Generally, these resource values are loaded into the server using a program called *xrdb* (the X resource database manager) and are accessed automatically when you run a client. Storing your preferences in the server with *xrdb* also allows you to run clients on multiple machines without maintaining a *.Xresources* file on each machine. Chapter 12, *Setting Resources*, describes resources in detail.

Some programs, such as window managers, have their own customization files. For example, there is a file for the FVWM window manager called *.fwmrc*, which is kept in your home directory. By editing the *.fwmrc* file, you can modify aspects of the window manager's operation, such as the contents of menus, and the key and pointer-button sequences used to invoke actions. Other window managers have comparable configuration files. See .the chapter for your window manager in Part II for more information.

Server Access Control

By design, X is network transparent. In theory, this means that a user can run clients on any host in the network and those clients can communicate with the user's server. However, there are security issues that need to be addressed and that constrain this concept of an open network. You certainly don't want any random user connecting to your X server. If you are setting up security for your system, you'll need more detailed information than is presented here.*

The forms of access control that X provides are:

Host-based
: Under host-based access control, the X server accepts connections only from a list of hosts specified in the file */etc/X0.hosts* or */etc/hosts.allow*; additional hosts can be added with the *xhost* client. Likewise, hosts specified in */etc/hosts.deny* are specifically denied access. When using the more secure mechanisms listed below, the host list is normally configured to be an empty list, so that only authorized programs can connect to the display.

User-based
: User-based access control verifies the user's identity before allowing access to the X server. All the methods listed here depend on the existence of a file called *.Xauthority* in the user's home directory. Identifying information is stored in the file to verify the user's authority to access the server. The mechanisms are:

  MIT-MAGIC-COOKIE-1
  : This scheme involves a special 128-bit code called a "magic cookie" that is known by the X server and is also made available to the user's account. Clients must present the magic cookie before they can access the server. Because the

* One starting point is the X security manpage. There are also several general Linux security HOWTOs available; check the Linux Documentation Project at *www.tldp.org* for current versions.

information is passed in plain text, this method of access control is vulnerable to snooping.

XDM-AUTHORIZATION-1
: This is similar to MIT-MAGIC-COOKIE-1, but the code is encrypted so that it can't be snooped as it is passed over the network.

SUN-DES-1
: This method uses Sun's Secure RPC--a secure public-key remote procedure call system.

MIT-KERBEROS-5
: Uses a password to obtain a *ticket* from a Kerberos server. Kerberos is a network-based authentication scheme developed by MIT for Project Athena.

The most common methods for restricting access to your server are host-based access control and the MIT-MAGIC-COOKIE-1 scheme. The XDM-AUTHORIZATION-1 and SUN-DES-1 are also user-based methods, but they are built upon DES (the Data Encryption Standard) and were long not exportable outside the U.S.