CMU-ITC-85-041

Unix Facilities for Andrew

David S. H. Rosenthal

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh PA 15213

Introduction

The software basis for the Information Technology Center's plans for campus-wide deployment of personal workstations is called Andrew. It consists of two major components; a remote file system and support for bitmap displays and mice as the user interface.

The Andrew system currently runs on a number of different types of workstations and mini-computers, all running the 4.2BSD version of Unix. Porting the system to another version of Unix would raise some issues; those that we can forsee are discussed here.

Prerequisites

Carnegie-Mellon has devoted much time, effort and investment to ensure that all significant computers on campus can communicate using the DARPA protocols. Although a version of Unix that supported some other protocol family, or no networking at all, might be technically capable of supporting at least the user interface of Andrew, it would not be acceptable to CMU.

Remote File System

Andrew's remote file system operates by caching whole files on a workstation's local disk. It places two requirements on the underlying Unix system:

1. It must be possible to add code to the kernel to intercept certain file system operations, and pass information describing them to a user-level cache manager process. The files affected in 4.2 are:

   ../h/errno.h
   ../h/file.h
   ../h/inode.h
   ../h/mount.h
   ../h/proc.h
   ../h/user.h
   ../sys/kern_descrip.c
   ../sys/kern_exec.c

../sys/kern_exit.c
../sys/kern_fork.c
../sys/kern_sig.c
../sys/quota_kern.c
../sys/sys_inode.c
../sys/ufs_inode.c
../sys/ufs_mount.c
../sys/ufs_nami.c
../sys/ufs_syscalls.c
../sys/ufs_xxx.c
../sys/vm_page.c
../machine/conf.c
../machine/machdep.c

Note that some important .h files are m,odified, so installing this intercept code requires that the entire kernel be re-compiled from the source.

2. The cache management process uses the socket mechanism to communicate with the remote file servers.

User Interface Support

The user interface support in Andrew consists of a user-level window-manager server process, that mediates all access to the display. Clients make remote procedure calls on this service over TCP/IP stream socket connections. Porting the user interface support to a new display involves changes only to the server process, and is normally only a few days work.

The Athena project at MIT has a similar window-maanger server process, called X. Although there are several differences between Andrew and X, we hope that X can be made upwards-compatible from Andrew. Their requirements for the underlying Unix system should be identical.

Socket Mechanism

The communication of the remote procedure calls between the clients and the server uses the socket mechanism of 4.2BSD, specifically stream sockets. An alternative inter-process communication facility that permitted unrelated processes to establish reliable bi-directional byte stream connections could be used, but would sacrifice remote access to windows.

Select System Call

The **select()** system call permits processes to wait for I/O activity on multiple descriptors. To support Andrew, it must at least work for descriptors representing the mouse, keyboard, pseudo-TTY, and IPC channels.

Non-blocking I/O

It must be possible to set at least those descriptors that support **select()** into non-blocking mode.

Pseudo-TTYs

Either the pseudo-TTY mechanism of 4.2BSD, or the stream I/O mechanism of Version 8 is required to support Andrew. These facilities permit processes to act as terminal emulators. Andrew uses the TIOCREMOTE mode of 4.2BSD's pseudo-TTYs.

Signal Mechanism

The signal mechanism of earlier versions of Unix suffers from race conditions, and is unreliable. Andrew requires reliable signals, such as those implemented in 4.2BSD.

Direct Access to Display

The user-level window manager must be able to map the actual pixels of the display, or for displays with autonomous RasterOp hardware the device registers, into its address space. Systems (such at the AT&T Unix PC) which forbid user-level access to the display, and require a system call per RasterOp, would not provide acceptable performance.

Mouse Driver

The Andrew window manager performs its own mouse tracking, accessing the mouse as a TTY-like device, and interpreting the hardware protocol for several mice. A kernel that supported mouse tracking would provide smoother movement, but it would probably be necessary to map the registers controlling cursor position into the window manager process while still providing **select()** indication of mouse movement. At times, Andrew uses relatively large (> 64 pixel) cursors.

Large Networks of Unix Workstations Need Auto-Configuration

David S. H. Rosenthal

Information Technology Center
Carnegie-Mellon University
Pittsburgh PA 15213

Introduction

A feature of some Unix kernels of particular importance for large networks of workstations is auto-configuration, the ability to determine the set of available peripherals at boot time. This paper discusses the reason for its importance, and some implications for hardware design.

The Problem

The Unix kernel is not, in general, a completely static piece of software, installed in ROM during manufacture. It changes as bugs are fixed and extensions are made. Nor are the workstations to be deployed on campus fixed hardware configurations. Options will be added or removed during their lifetime.

The System V kernel must be configured for the set of devices actually present on the machine it is to run on. Attempting to use a device that is not actually present is a fatal error; there is no way for application software to determine whether a configured-in hardware device is actually present in a safe way.

In a large network of workstations, many of them owed by individuals, it will be extremely difficult to:

a) Keep track of the actual hardware configurations of each machine.

b) Ensure that each machine acquires the appropriately configured kernel whenever its hardware is changed.

c) Ensure that fixes to the kernel get propogated to individual machines in a timely fashion, and in the appropriate configurations.

Auto-Configuration

The Berkeley kernel, on the other hand, auto-configures itself. Attempts by application programs to use devices that don't exist are quite safe; the `open()` call on the device file fails. All workstations of a particular type (for example all our Suns) can run the identical binary image of the kernel. At boot time, this kernel examines the hardware it is running on to determine the set of devices available.

Thus no effort is required to track hardware configurations; bringing up a new or modified workstation merely requires use of the current binary kernel image. Similarly, only one new image need be propogated when a fix is made.

Hardware Requirements

4.x BSD Unix auto-configures by calling a `probe()` routine for each device that it supports. The responsibility of the `probe()` routine is to locate all instances of its device that are actually present in this particular machine. It does so using a compiled-in list of the possible I/O locations at which instances of the device may exist, and attempting to verify their presence. Different devices may share I/O locations provided it is possible for software to disambiguate them (for example, by reading registers or causing interrupts to different vectors).

This process can be performed only if the hardware:

a) Permits the presence (or absence) of a device at a particular location to be determined,

b) Permits the parameters of different storage media attached to the same controller to be determined (for example, it must be possible to tell how many sectors, heads and cylinders a particular disk has),

c) Permits devices sharing the same I/O location to be disambiguated.