Chapter 1 Introduction

The X Window system, or more simply “X” or “X11”, is the de facto standard graphical engine for UNIX-based operating systems (including Linux), and provides the only common windowing environment bridging heterogeneous platforms in today’s enterprise computing systems [1]. Thousands of independent software developers provide applications that run on the X Window system, and the worldwide community of users of the X Window system currently exceeds 30 million.

One of the strengths of the X Window system is that it supports the thin-client/server computing model, which consists of three key components (Figure 1): (1) thin-client hardware devices (known as X-terminals in the X Window system), (2) the application server, and (3) a display protocol. All the applications and data are deployed, managed, supported, and executed at the application server. Thin-client devices gather inputs from users in the form of mouse clicks and keystrokes, send them to the application server for processing, and collect screen updates as the response from the application server. All the interactions between thin-client devices and the application server occur via an efficient display protocol.
Unfortunately, the display protocol of the X Window system does not support the transmission of audio data. In a UNIX-based operating system, the standard way for application programs to output sampled audio data to an audio hardware device is to invoke a sequence of open, ioctl, write, and close system calls to some audio device files such as “/dev/audio” and “/dev/dsp”. These system calls can only deliver audio data to the local audio hardware device – they cannot deliver audio data to an audio hardware device in another computer. Thus, all the programs executing in the application server cannot use the standard method of delivering audio data to a thin-client device.

Several methods have been proposed for overcoming the lack of network audio functionality in the X Window system and for implementing audio applications in UNIX-based operating systems [2,3,4,5]. One of the solutions is to define and provide a network audio library [2,3], whereby all the programs (or processes) that want to
send audio data over the network must invoke subroutines in this library. This solution can be applied to a multiuser environment since the network audio library will arrange a separate channel to transmit audio data for each user. However, programs that use the standard way to send audio data (i.e., invoking a sequences of open, ioctl, write, and close system calls) cannot work in this kind of environment. Instead, these programs must be redesigned to invoke the subroutines in the new network audio library.

Another drawback of this scheme is that the program cannot use any audio data format or device types that are not defined in the network audio library.

The BNAS (basic network audio system) proposed by Innocenti [4] does not rely on a new network audio library, removing the need to modify or recompile existing programs. Instead, it uses a FIFO special file to redirect the audio data sent to “/dev/audio”. However, the BNAS cannot support multiuser operation, so that all the audio data delivered by the programs of different users must be forwarded to the same destination. Also, the original audio hardware device in the application server must be turned off during the operation of BNAS.

In this paper, we propose a multiuser network audio system called MuNAS which can work in any UNIX-based operating system. First, it is designed to work on a multiuser operating system – several users can activate the network audio system by executing their own audio applications in the same computer, with their audio data
capable of being forwarded to different computers simultaneously. Second, existing audio applications do not need to be modified or recompiled, and no additional libraries are required. Third, the program is not restricted to any special form of audio data format or device type; i.e., the audio applications can arbitrarily define their own audio data formats and device types.

The implementation of the MuNAS comprises embedding a system-call serializer in the kernel of the operating system, making some audio data relays, and executing daemon processes in the audio-sending and audio-receiving computers. An audio data relay can be either a FIFO special file, a message queue of interprocess communication, or similar mechanisms. We present two ways to implement the system-call serializer: one is to write a new device driver for sound device and the other is to modify several system calls in the kernel. We have implemented MuNAS in Red Hat Linux 7.2 (kernel version 2.4.7-10), 8.0 (kernel version 2.4.18-14) and 9.0 (kernel version 2.4.20-8) [6], and have tested the implemented system using mpg123, gtv, mplayer, realplayer, alsaplayer, shockwave flash player, record, and Esound. The MuNAS performs well, as indicated by its ability to send stereo, 16-bit, 44.1-kHz linear audio data to several users in the same Linux PC simultaneously. According to our experiment, the MuNAS is compatible to the Open Sound System [7,8]. The scheme proposed in this paper can be applied to any UNIX-based operating system.
The remainder of this paper is organized as follows: Section 2 surveys the previous work in network audio system, Section 3 gives an overview of basic audio programming in UNIX, Section 4 presents the architecture and implementation of the MuNAS proposed in this paper, Section 5 is our implementation details, Section 6 is future work and conclusions are provided in Section 7.