Semester Thesis

Building a client/server multimedia-kiosk system using pxe, root-over-nfs, mozilla and a CMS
a.k.a Multimedia Kiosk revisited

Mathias Payer

Martin Probst
Adviser
Prof. Dr. Bradley J. Nelson
Institute of Robotics and Intelligent Systems
Swiss Federal Institute of Technology Zurich (ETH)

2005-06

IRIS
Institute of Robotics and Intelligent Systems

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

In this semester project we tried to develop a multimedia kiosk system that is easy to configure, easy to maintain and easy to use. We used a central server and multiple diskless ("dumb") clients. These clients load all data from the central server using techniques like PXE and TFTP to bootstrap the system, NFS to transport data as well as programs, a HTTP interface and a browser to display the content to the users in front of the kiosk.

The content can be changed by a user friendly CMS and the kiosks are configured on the central server by no more than 4 configuration files. Maintainance of the clients and the server can be done on the central server. In order to reduce the risk of this single point of failure a backup strategy and tight security is enforced.

Contents

Chapter 1

Introduction

The Institute of Robotics and Intelligent Systems (IRIS[1]) is evaluating new technologies to replace the way information is provided to students. Currently, large scale printouts are distributed to different locations all over the ETH campus. These printouts will be replaced by "Multimedia Kiosks"[2].

All these kiosks will share the same (unix) [11] operating system and perform a diskless [4] boot from a central server. After the boot process has completed, the kiosks start a browser and display a webapplication that was especially developped for touch-screen use.

One of the goals of this semester thesis was to make the process of adding new kiosk clients as easy as possible. So the only changes that have to be made are the ip-address configuration, the X-Server configuration and the base-url that is loaded right after the startup has completed.

A positive side-effect of this layout is that the configuration is handled locally on the server and the kiosks themselves carry no configuration whatsoever. Using this paradigm we can build cheap and small kiosks without any moving or rotating parts like hard disks.

Another goal was to change and distribute information as easily as possible. In oder to accomplish this a Content Management System (CMS) was used. A CMS can easily be handled by non-technical personnel. This gives us the advantage that no highly skilled personnel has to be used to keep the kiosks up and running. It will be easy to adjust the homepages and presentation pages of the different institutes using an intuitive web interface.

One other aspect was to offer the possibility to add kiosks for other institutes. This can easily be done with the base-url configuration option which can be set for each kiosk. The CMS also offers the possibility to integrate different institutes on the same server.

Throughout the document some technical terms like DHCP, PXE, Syslinux and so on will be heavily used. In oder to make the definitions easier, some doc-

umentation about these techniques will be offered in the "Preliminaries" chapter. Additionally some of these abbreviations will be described in the glossary.

After the preliminaries, some related work that offers good starting points will be described. We will present some existing howtos and information how it is possible to boot diskless clients. Then we will talk about other kiosk systems and their problems and drawbacks.

Then we present the concept and general layout. This section describes how we implemented the kiosk system and what our design criterias were. After the concept is shown, the implementation will be presented. We will go through all the relevant server and client side steps to setup and configure a running kiosk system.

At last we will discuss some of the important problems we encountered, such as the PXE boot problem. There will also be some details about limitations and drawbacks of our solution.

Chapter 2

Preliminaries

This chapter will provide some information that will be used throughout the whole thesis. Here we will describe the basic techniques that are needed to build a kiosk system.

Some details about the boot process, the configuration of the network devices and the storage system will be explained. Later on in this chapter the Linux-kernel configuration will be discussed as well as the chosen CMS that is used as the kiosk frontend.

All the described network services build upon the internet protocol (IP, RFC 791), the transmission control protocol (TCP, RFC 793) and the user datagram protocol (UDP, RFC 768)

2.1 DHCP - Dynamic Host Configuration Protocol

This is a client-server networking protocol, used to remotly configure clients. They need no prior knowledge about the network structure. Everything these clients need to know is provided by the responding DHCP server.

When a client tries to configure a network device, it only knows its MAC address (media access control address). Using this MAC address the client sends a DHCP discover via multicast to all devices in the same subnet. Now the DHCP server (or multiple servers) can respond to this request by a DHCP offer to the client (this is sent via unicast). The client then selects one of these offers by sending a DHCP request to the server. If the client has some state and knows its last address, then it will try to select the last used IP-address. The server then responds by a DHCP acknowledge.

Using this protocol the client registers an IP address for a given lease time. After the time is up, the DHCP registering process needs to be redone. Additionally, the server can supply a lot of intormation to the client, like gateway and DNS configuration and the TFTP server that offers a boot-image.

2.2 PXE - Preboot Execution Environment

PXE (published by Intel) is an environment to bootstrap stateless computers. For example diskless computers, so no hard-disk or pre-installed operating system needs to be available to boot the client. To start a client, PXE needs a network card to access a server.

Many newer network cards are shipped with a PXE bootrom or at least with a socket for a PXE bootrom. After the bootup tests are completed the BIOS gives control to the PXE rom, instead of the hard disk. The PXE rom then issues a DHCP request and waits for an DHCP offer with an URL to a small boot-image as special option.

The PXE rom then tries to download the specified bootstrap program via TFTP into the local RAM and gives control to the downloaded image.

2.3 Syslinux / PXELinux

Syslinux is a set of bootloaders (bootstrap programs) that will load a Linux kernel into memory and then give control to the kernel. In our case we use PXELinux. This is a small image that can be loaded via the PXE rom from a network card. We do not need any local storage as the bootstrapping code is transmitted over ethernet.

By using PXELinux you can specify the boot options and kernel options on the server. After the PXELinux loader is started, it tries to download a predefined kernel from the server via TFTP into the memory of the client.

One can compare PXELinux to bootloaders like Grub or Lilo. The only difference is that Grub and Lilo load the Linux kernel from a local hard disk, PXELinux loads the kernel from an TFTP server.

After the download is complete, it starts the kernel and Linux will boot the computer into an usable state, mount the root filesystem and configure attached devices.

2.4 TFTP - Trivial File Transfer Protocol

TFTP is a very trivial file transfer protocol akin to a basic version of FTP. This protocol is mostly used to transfer small files between hosts on the local network. Differences between FTP and TFTP are that TFTP uses UDP, which does not offer reliable data streams as transport protocol. Additionally, the exact location of the file needs to be known because TFTP does not transfer file meta information like folders. This protocol has no authentication mechanism to make it possible to boot

diskless clients. The only usage of TFTP is to transport files from the server to the client.

This is exactly what is needed to transfer bootloaders, bootimages and kernels from a central server to a diskless client.

2.5 NFS - Network File System

The NFS protocol (invented by Sun Microsystems) makes it possible to export filesystems using the internet. Clients can mount these filesystems and perform updates (reads and writes). The server uses RPC (remote procedure calls) to allow client interactions. The NFS server itself is divided into several parts. The first part a client encounters is the portmapper. This service maps RPC-functions that a remote client can call to TCP or UDP ports. When the client knows the right port, it contacts the mount service. This service checks access rights and permissions of the client and then grants (or does not grant) access to the data. From this point on the client has mounted a remote filesystem on his local system and can then access data on the remote server.

NFS supports multiple ways to limit access to the exported filesystem. The first rudimentary option is to filter for ip-addresses. Then one can either use a unix group and user id (this is considered insecure) or credentials from a kerberos server. This is a secure server that handles encrypted logins and credentials.

2.6 Kernel configuration

A kernel is the most basic part of an operating system. Using Linux one can compile his own kernel with only the really needed options. So only the drivers that are really needed are selected and installed.

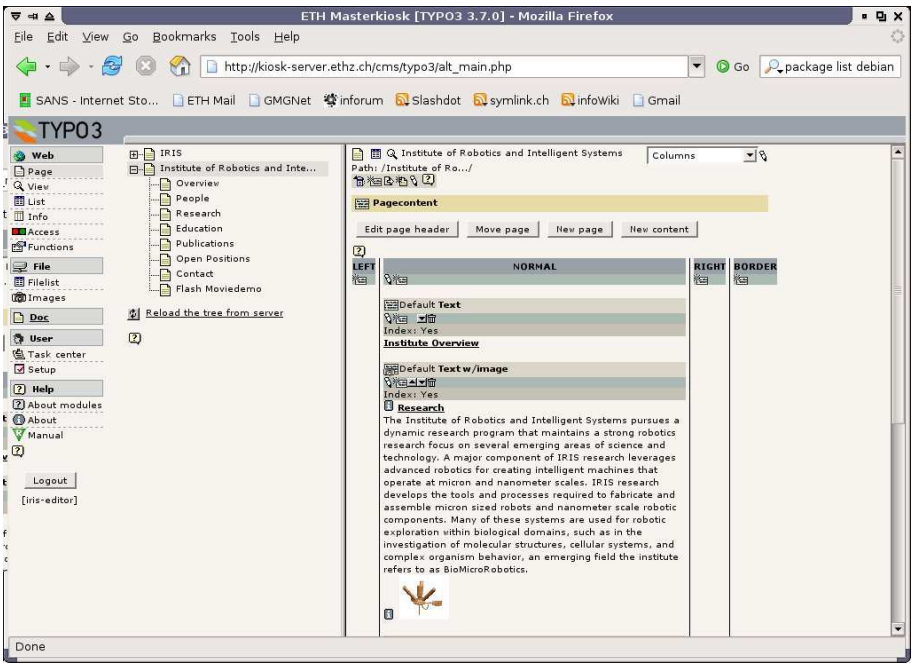
The basic process to build a specific kernel is to download the sources from "http://www.kernel.org" and untar them to "/usr/src". Then one enters the directory of the source-tree and configures the kernel with "make menuconfig". This command displays a screen where one can set all the options and drivers.

After the configuration is done, the kernel needs to be compiled. To simplify the compilation process the debian-tool[10] "make-kpkg" out of the kernel package is used. The following command will generate a kernel package: "make-kpkg –bzImage –revision kiosk0.1 binary-arch modules_image".

2.7 Typo 3 - CMS

Typo3 is a content management system that is able to handle multiple webpages and multiple users. It is easy to add new users (and groups) and to give them different access rights. So the pages of multiple kiosks can be administered easily.

One of the biggest advantages of a CMS with a web interface is that the design and content editing is easy and can be done by non-technical personnel. There are also many good tutorials which describe how to work with this CMS.



Chapter 3

Related work

In this chapter some information about related work will be presented. We will present how this information is used in the kiosk project and where to find more details about these projects.

3.1 Root over NFS

This is a special kernel option that makes it possible to mount the root file system over a network. Using this option a client does no longer need any attached storage (as long as the kernel is already loaded). Please see the kernel documentation[13] for more details.

A drawback of this technique is tha only limited documentation exists. One has to search a lot of email archives and read many howtos to gather enough knowledge to configure a root over NFS system.

3.2 PXE Boot howto

Reading the diskless Linux with PXE boot howto[14], one gets the general impression how a kiosk can be booted without local storage and configuration. This howto talks about installing TFTP, DHCP and NFS.

3.3 Adapted mozilla for kiosk use

There are already about two different mozilla projects that try to adapt for kiosk use[15]. They display only rudimentary controls (like home, back, forward and stop) and one big browser window. The existing solutions have two disadvantages that prohibit their use:

1. Their design is suitable for home and personal use, but not for a university (funny images as buttons and so on).

2. Most solutions also display a URL field and a touch keyboard where the user can change the address. On our kiosk the user should only be able to follow links and not be able to enter its own addresses.

3.4 Other Kiosk systems

There are already some existing kiosk implementations and howtos. But the problem of these implementations is that they are very limited in their functionality. Most of the kiosks only work with local storage and offer limited web support.

After a detailed study of the available systems, we concluded to build our kiosk system from scratch. This way we could realise all our ideas about an optimal multimedia kiosk. The most dominant of these ideas are diskless clients, a global CMS with user management, a central configuration, security and high availability.

3.5 Old multimedia kiosk

There already exists another semester project that implemented a multimedia kiosk. This project also used a VIA Mini-ITX mainboard and had a TFT touchscreen attached. One of the main differences is that a SuSE Linux distribution was used, which is fairly bloated compared to Debian and relied on a local harddisc. Additionally there was no central configuration and every kiosk would work for itself and had to be installed, configured and maintained.

Other differences are that we use a different (adapted) browser and a special window environment that ensures that kiosk specific demands are met (examples are auto login, restart of the window environment if the browser fails, remote administration login and more).

One of the biggest differences is that the new kiosk system uses a central server with a CMS. The old kiosk displayed a normal webpage (the homepage of the IRIS group), now we have special pages that are designed for kiosks. A powerful content management system was developed that is ready for multiple institutes.

Chapter 4

Concept

This chapter will provide detailed information about the chosen design. The first and second parts will talk about hardware issues. We will show a possibility to raise security and to ease maintainability. At last we will talk about extensibility to multiple institutes, users and kiosks.

4.1 Hardware concept

The basic concept of the multimedia kiosk system is to use a master server and many (dumb) kiosk clients that load everything from the server. So all data is stored in a central, easy to manage place and the clients do not need any local harddrives.

Using this solution there are serveral advantages to the decentralised version. One of the biggest advantages is that all clients can be configured in a central place. The clients also share most of the file system (programs, libraries, data) and only a very small hardware dependent part (kernel, touchscreen drivers and some configuration files) differs from kiosk to kiosk.

Another benefit is that backups can be done in one central place and the kiosks merely need configuration. It is also quite easy to add a new kiosk, no installation is necessary and if the hardware is close to other, already running clients, the new kiosk will be up and running in a few minutes.

As the clients do not have any local storage the downtime will be much lower compared to traditional systems, where the harddrive is one of the parts that fails most of the times.

4.2 Software concept

On the server and on the kiosks we use a Linux based operating system. We do this out of serveral reasons. The most dominant ones are stability and adaptability.

Using a Linux distribution we can tweak startup scripts and adjust everything to suit our needs.

Our kiosk does not need write access to the filesystem. So we adjusted the Debian distribution to exclude write access in the startup scripts. This way we could easily increase the security of the central server.

One important idea was that the kiosks should be very easy to administrate and the actual machines should be very easy to setup. An advantage of our solution is that we do not need to configure the clients (we only need to set some BIOS settings).

4.3 Extensibility to multiple institutes, users and clients

One of the main drawbacks of the old kiosk system was that it was an isolated application. To get a new kiosk (for another or the same institute) up and running, one had to go through the whole configuration process. The new kiosk system should avoid this and provide simple expandability to add new clients, to handle the web pages and to configure the server for new users and institutes.

Our kiosk system uses a content management system that is easy expandable to new institutes. One can easily add a new user and new pages that will be served on a new kiosk for some other institute. Everything should be as expandable as possible, but still quite easy to configure and maintain.

Chapter 5

Implementation

In this chapter we will discuss design criteria and decisions about the used architecture. First the server side view will be described including the running services and the exported filesystems and structures. Then we will discuss how the client is able to boot and get a usable configuration without any local information except the MAC id of the network controller. Finally the kiosk CMS will be presented. Using this CMS the viewable information on all the kiosks can be adjusted.

5.1 Server side

Here we will describe the running services on the server and what kind of functions they provide. The second part of this section will show the filesystem structure on the server that holds all the configuration and data files for the different clients.

5.1.1 Services for clients

The server needs to provide many different services to let the client complete the different boot stages. First of all the PXE ROM on the client asks via DHCP for an IP address. Then the PXE bootloader and the Linux kernel are transfered via TFTP. Now the kernel on the client loads the filesystem via NFS and finishes bootup. After the browser is started on the client, all pages are served via the HTTP service on the client.

DHCP The DHCP service handles only the kiosks (it filters the incoming DHCP requests by MAC addresses) and passes them the required information, such as IP address, DNS servers (to map between names like www.ethz.ch and IP addresses) and the TFTP server where the kiosks can find the PXE environment.

TFTP This service is kept as trivial as possible. It is only able to serve small files, like a bootloader or a Linux kernel.

NFS The NFS service exports the root filesystem and the kiosks' configurations. All kiosks should share the same read only filesystem. Only a few files will be different per kiosk, like hostname, IP address, X-Server configuration (different graphiccards), Linux kernel (different kiosk-hardware) and start URL (the URL of the first page that will be displayed after the bootup is complete).

HTTP We will use Apache as an HTTP server. After the bootup is complete the kiosk starts displaying webpages. These pages come from the central kiosk-server. This makes it possible to maintain all the webpages on a central server. This has the advantage that the information is not spread over multiple servers and eases updates, changes and maintainability.

Database As we are using an advanced content management system we save our webpages not as static pages on a harddisc but the pages are constructed dynamically from a database.

Backup Because we only have a central server, backups are inevitable. So important data like kiosk configurations, database dumps and system configuration must be backuped on a regular basis. Of course the server storage must be secured by a RAID (Redundant Array of Inexpensive Disks) to circumvent data loss.

5.1.2 Serverside filesystem structure

In the early stages of the kiosk project we already designed a special filesystem structure to handle multiple clients. All the clients share a central root directory and configuration. This shared configuration is a "normal" Debian system, that resides in a subdirectory of the server ("/home/kiosks/default"). Every client then has some additional specialised configuration files:

1. Every client has a special entry in the DHCP server configuration that assigns an IP address and a domain name to a client (based on its MAC address).

2. Also based on the MAC address every clients gets its own Linux kernel via TFTP (if multiple clients share the same hardware configuration, then they can share a kernel via symbolic links).

3. After the Linux kernel has booted, the startup scripts are executed. As hardware and configuration of the kiosks may differ, some files change between hardware revisions. To configure a separte start URL (that is displayed when

the kiosk has started) and XFree configuration (for the graphical environment, parameters like graphic card, touchscreen and so on are configured there) every kiosk has its own directory (the selection is based on the kiosks IP address). To ease access to these configuration files a symbolic link was made in the kiosk home directory.

5.2 Client side

In this section we will provide information how the kiosks use the services on the server. We will discuss the bootup process, how data is accessed and how the kiosks can work without any writeable media (neither local nor remote).

5.2.1 DHCP – Get IP and TFTP-Server

When a client is started it first loads the BIOS and does a POST. After the hardware passed these two stages, the BIOS loads the PXE environment from the network controller and executes it. This PXE environment will query the LAN for a DHCP server, an IP address and a TFTP boot server.

Now that TCP/IP settings are configured, the PXE environment asks the TFTP server for a bootloader and downloads this bootloader into memory. After the transmission has sucessfully completed this boatloader is executed.

5.2.2 PXE Boot – Syslinux

We use Syslinux as bootloader that is sent to the kiosks. This boot environment is very small and covers all needed features.

When the Syslinux bootloader is started, it asks the TFTP server for a special configuration file. This file is comparable to a lilo (Linux loader) configuration. It states the name of different kernels and a default kernel. Additionally we can specify some special kernel parameters in this file. For our kiosks we tell the kernel to configure itself upon boot via DHCP and to mount the root filesystem. After a short timeout the default kernel will be downloaded into memory from the TFTP server. If the download completes successfully, then the bootloader will execute the kernel.

5.2.3 Mount the root filesystem via NFS

When the kernel is started it tries to initialise all hardware. After this is done it tries to mount the root filesystem. Then the kernel will execute the bootup script of the distribution that will start all services.

Normally, the root filesystem is on the local machine. To make it possible for the kernel to mount the root filesystem from a remote computer, the TCP/IP system first needs to be working. Thats why there exists a kernel option that does kernel level autoconfiguration of the networking subsystem. After this subsystem is initialised (via DHCP), the kernel is able to mount the remote root filesystem and can then execute the startup scripts from the remote server.

5.2.4 Limitation to read-only root filesystem

Out of security reasons we wanted a read only filesystem on the kiosks. The clients should not be able to change any data on the server. Another reason for this limitation is that NFS traffic is unencrypted and unsecured and can easily be intercepted. So an attacker could easily gain write access to the server if he is able to catch any network traffic between the server and a kiosk.

Normally a Linux distribution needs write access to some parts of the filesystem (for temporary files in "/tmp" and "/var/tmp", lock and pid files in "/var/run", log files and many more). Mozilla also needs write access to its profile (for lock files and to store data in the cache). So we made a custom startup script that initialises a temporary RAM disk with all the needed folders and files. We do not need these files from our clients, so we do not care that they are lost after a reboot.

5.2.5 Touchscreen configuration

The touchscreens we used for our demo kiosks are both from ELO systems. One of them is controlled by a USB interface and the other by a serial interface. So we activated in our kiosk kernel support for USB and serial devices. This is enough for these ELO devices. Additionally a special driver is selected in the X11 configuration that handles the touchscreens as input devices. After calibration is done, no further configuration is neccessary.

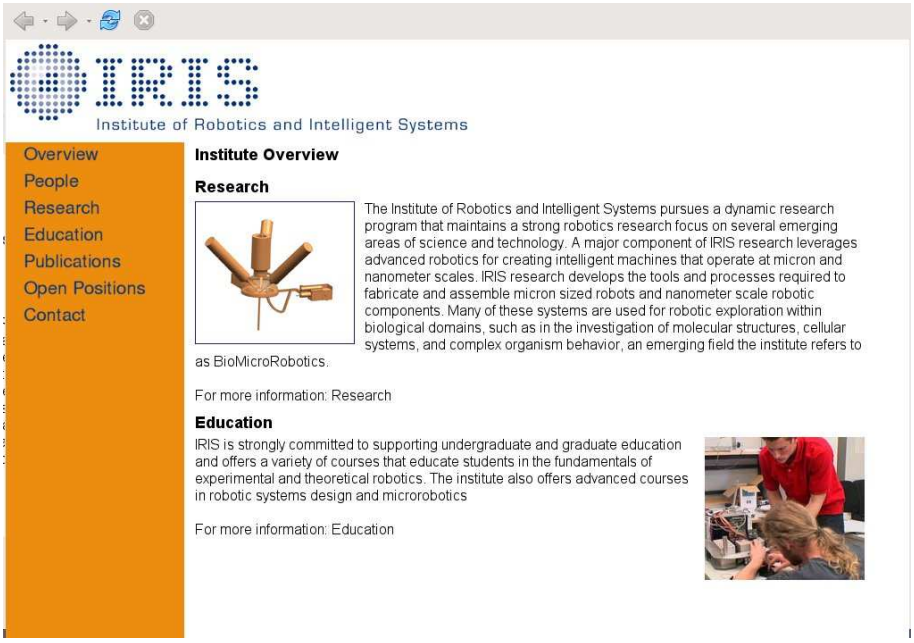
For some other types of touchscreens there exist special kernel drivers which first need to be compiled and acivated. One can do this during the kernel configuration.

5.2.6 Mozilla Firefox adaption

To suit the special kiosk needs we had to adapt a browser. After some testing and considerations we decided to use the Mozilla Firefox. This browser configures the user interface using some editable XML files. So we unpacked these definitions and adjusted them to our needs. We deleted the location and removed some unneded buttons and the menu.

These special definitions can be found in the java archive "/usr/lib/mozilla-firefox/chrome/browser.jar". One needs to complete serveral steps to change the user interface.

1. First one has to start the browser as a normal user.
2. Customize Firefox by right clicking beneath the menu bar.
3. Then delete all the unneeded menus and user interface components.
4. Now enter fullscreen by pressing F11 and then close Firefox. The only problem that we still have is that the close button is still displayed. We remove it by change the UI layout.
5. The file "browser.jar" needs to be copied to a temporary directory.
6. Then it must be unpacked with the "unzip" utility.
7. Now the file "./content/browser/browser.xul" can be edited and unneeded parts of the user interface (like close and minimize buttons) can be commented out.



To enable flashmovie playback one has to install the Macromedia Flash Plugin. The Linux download is available from the Macromedia Homepage. The plugin can then be installed into the "/usr/lib/mozilla-firefox" directory.

5.3 Kiosk-CMS

To provide all features that we presented in the concept chapter, we used the existing content management system Typo3. This CMS is already very advanced. It supports

a lot of features like multiple users, different pages, easy editing via a web interface and an administration interface. One big difference to other CMS is that content and layout are separated. First one chooses the layout for a page and then content entering is possible. The system is very flexible so that the layout can still be changed after text and data were supplied.

The setup and configuration of Typo3 is very easy. One has to download the package from the net and install it locally in "/var/www". Then one has to check if all the needed software and libraries are installed (a list of installed packages is included in the appendix) and a MySQL database must be set up. Finally, one enters the configuration page and sets all the needed configuration like administrator password and database connectivity.

Now the CMS can be configured, new users can be set up, layouts can be chosen and adapted and new pages can be created.

5.4 Hardware

To keep costs low and to ease exchangeability standard PC components where used wherever possible. The server is implemented on a Dell computer with no special features. The kiosks share Intel compatible mainboards and processors.

The only special parts are the TFT touchscreens and the cases for the kiosks. Different suppliers were evaluated and Jaeger was able to delivier a case with special dimensions that suited our needs. To secure our touchscreens from theft we constructed a special case that holds the screen and makes it hard to open the screws.

Chapter 6

Discussion

In this chapter we will provide some information about the problems we discovered during the implementation. We will also talk about other possible ways of solving special problems and drawbacks that our solution may have.

During the implementation phase some problems were discovered and solved. We will provide some information about the PXE boot problem that some VIA mainboards seem to have. Then we will describe limits of the DHCP configuration. Specially the limit that a kiosk needs access to a local DHCP server.

An advantage of our solution is that one can maintain the software versions and distribution of all kiosks at once. We will show how to update the software in an easy way and how to maintain the server. As the kiosk server is a single point of failure in our configuration we will show how to secure this bottleneck as much as possible.

Then we will discuss the content management system, especially how the information is organised and grouped. Additionally we will talk about the backup procedure of the server and the CMS.

At last a general overview about the running systems and their status will be given.

6.1 Implementation problems

First of all very few hardware related problems were encountered in this semester project. The difficulties started with the implementation and interaction of the different used services. Most of them could be sovled by reading the documentation or using a search engine to find information from somebody who had similar problems.

6.1.1 PXE Powerdown/-up problem

One of the main hardware problems was a PXE boot issue with some VIA mainboards. From time to time these mainbaords refused to boot. They started up and completed the POST without problems. Then they loaded the PXE code from the ethernet card and then failed to get an IP. The first assumption was a misconfiguration of the DHCP server, but network flow analysis showed that the kiosk did not send out any packets.

So the problem must reside in the BIOS configuration. We tried almost all settings but we could not discover the real root of this problem. Then we discovered that the problem emerged every time we used APM (Advanced Power Management) calls to power down the machine. The Linux kernel uses special APM calls to completely power down the computer and to turn off the power. After we halted the machine using these APM commands it was no longer possible to start the kiosk. The only solution is to remove power from the mainboard for about 10 to 20 seconds (until all the capacitors run out of power). The problem was that if a computer is turned off, the mainboard is still running. It looks as if this APM call crashed the running BIOS and when the mainboard was turned back on the crashed BIOS failed to load the PXE environment.

The solution to this problem is to cut power from the kiosk and not to use the normal shutdown scripts. As we do not have any writable filesystems we will not lose any data. And shutdown is even faster.

6.1.2 DHCP limitations

When we configured the kiosk DHCP service we had to make sure that it does interact nicely with the existing one. The problem was that the existing service should not give IP addresses to our kiosks as this server does not know anything about our bootservers and our service should not handle IP addresses to computers that are not kiosks.

Currently we solved this problem using a filter on the MAC addresses of the ethernet cards. Each service only serves these computers with well known MAC addresses. One possible solution for the future would be to merge these two services and send additional data (like the bootserver) to the kiosks.

One other limitation of the DHCP protocol is that it is only routed over one hop. This means that DHCP requests can not pass router barriers. So we have technically a limitation that every subnet needs its own DHCP server. But this should not be a problem. When we want to place kiosks somewhere on the ETH campus we have to keep in mind that we need to configure the local DHCP server to send boot parameters to our kiosk (the kiosk systems can easily be filtered by

MAC address). With these boot parameters the kiosk can contact our (remote) boot server.

6.2 Update procedure

One of the biggest advantages of our solution is the easy update procedure of the kiosks. All kiosks can be updated at once. So we have to keep two machines up to date. The kiosk server (where we will need tight security) and all the kiosks summarised as one machine.

The kiosk server is currently configured so that only as few services as possible are running. These services can be updated using the Debian way of system maintainance. One must login as root to the server and then issue "apt-get update && apt-get upgrade" to upgrade the kiosk server to the newest software versions.

The kiosks are a little more difficult. One first has to login to the server and then to change the root directory to the kiosk directory by issuing "chroot /home/kiosks/default". Then one can use the familiar command "apt-get update && apt-get upgrade" to upgrade all kiosks in an easy way.

These updates should be done about once in a week. They can either be started by hand or by a small cron script.

Listing 6.1: Eaxample for a cron update-script (/usr/sbin/security-update.sh)

```
#!/bin/bash
# Daily Security Updates
# root gets an EMail if new updates are available.
# These updates must then be installed by root.
#
# To run this script daily use the following
# entry in /etc/crontab:
# 53 6    * * *   root    /usr/sbin/security-update.sh
/usr/bin/apt-get update > /dev/null
UPDATES=$(/usr/bin/apt-get -qq --simulate upgrade | \
      /usr/bin/wc --lines)

# Print the available packages to stdout
if test $UPDATES -gt 0; then
        /bin/uname -a
        /usr/bin/apt-get -q --simulate upgrade
        /usr/bin/apt-get clean
fi
```

6.3 CMS Usage

The Typo3 CMS we used in this kiosk project is very intuitive. Most steps are self-explanatory or can be looked up in the online help.

Another good ressource to start with is the Typo3 how to [12]. Additionally, there is a lot of information available on the Typo3 website, including many good screen shots and tutorials.

The advantage of this CMS is that new users can be easily added. These users can then be assigned to institutes and are only able to change the institutes pages. Every institute gets its own homepage and an overview page on the central web page where all the institutes are summarised.

6.4 Backups

If this system goes into productive use a good backup strategy must be deployed. Currently we used a test system for our kiosk server. In the future the server should run on real server hardware.

We are also aware that our kiosk server is a single point of failure. All data and configurations reside on a single server. So we recommend daily backups of the database and weekly images of the complete server configuration (including the kiosk clients).

This backups can easily be achieved by cron jobs over night. The database backup has to save a specific directory and transport it to an external backup server. Images can easily be prepared by other scripts and the tar utility. Later on these images can also be transferred to the external server.

If these backups are done as proposed, a maximum of one days work on the kiosk homepages can be lost. And if some hardware fails, downtime will be as short as possible because we will already have images to reinstall the complete server in less than one hour.

6.5 Status

Today the server is configured to handle multiple clients. Additionally we have two kiosks running from this central server. The CMS contains a demo webpage for the IRIS institute.

The kiosk server is configured and running. One can easily add kiosk clients as stated in this documentation. The maintainance time of the server and clients should be as low as possible, the only disadvantage being the server as single point

of failure. This disadvantage can be limited by doing daily backups and securing this server as much as possible.

All the desired features are implemented. The kiosks can easily be added and configured. The home pages of the kiosks are managed over a central CMS and media and movies can be played by flash movies and flash applets.

Chapter 7

Appendix

7.1 Configuration files

In this section we will list all the important configuration files. These include all the details that are needed to replicate the currently running solution.

7.1.1 Server side

Here we provide all the server side configuration, especially all the running daemons and services.

7.1.1.1 DHCP-Server configuration (/etc/dhcpd.conf)

On the kiosk-server we have a running DHCP service to assign IPs, DNS servers and TFTP server to the kiosks. These configurations can be seen in the following file.

Listing 7.1: DHCP service configuration file (/etc/dhcpd.conf)

```
# Kiosk-DHCP configuration
# May 05 / Mathias Payer (payerm@student.ethz.ch)

# Default domain name
option domain-name "ethz.ch";
# Some eth name-servers
# The kiosks need these servers as they have to resolve
# the kiosk-server.ethz.ch entry to get their boot images
option domain-name-servers 129.132.98.2, 129.132.250.2;

# some default values
option subnet-mask 255.255.255.0;
```

```
default-lease-time 600;
max-lease-time 7200;
deny unknown-clients;

# "our" subnet, where the all the kiosks are in
# but we don't give out dhcp-leases to other hosts
# than kiosk serves
subnet 129.132.113.0 netmask 255.255.255.0 {
        range 129.132.113.77 129.132.113.77;
        option broadcast-address 129.132.113.255;
        option routers 129.132.113.1;
}

# first kiosk-host, identified by its ethernet address
host kiosk01.ethz.ch {
        # allow boot via ethernet
        allow booting;
        allow bootp;
        hardware ethernet 00:40:63:D5:12:DE;
        fixed-address 129.132.113.77;
        option host-name "kiosk01.ethz.ch";
        # bootloader (that we get via tftpd)
        filename "/home/kiosks/tftpboot/pxelinux.0";
        # server to get the tftp images from
        next-server kiosk-server.ethz.ch;
}

# second kiosk-host, identified by its ethernet address
host kiosk02.ethz.ch {
        # allow boot via ethernet
        allow booting;
        allow bootp;
        hardware ethernet 00:40:63:DC:EE:4C;
        fixed-address 129.132.113.74;
        option host-name "kiosk02.ethz.ch";
        # bootloader (that we get via tftpd)
        filename "/home/kiosks/tftpboot/pxelinux.0";
        # server to get the tftp images from
        next-server kiosk-server.ethz.ch;
```

}

7.1.1.2 TFTP configuration

The TFTP service is very small. As this he does not have it's own configuration-file, but is started from the internet daemon, the inetd. The relevant configuration-line in "etc/inetd.conf" follows:

Listing 7.2: Bootloader configuration

```
tftp dgram udp waitnobody /usr/sbin/tcpd \
  /usr/sbin/in.tftpd --tftpd-timeout 300 --retry-timeout 5 \
  --mcast-port 1758 --mcast-addr 239.239.239.0-255 \
  --mcast-ttl 1 --maxthread 100 --verbose=5 \
  /home/kiosks/tftpboot
```

7.1.1.3 NFS configuration (/etc/exports)

To export files to the clients we use the NFS service. This daemon needs hardly any configuration. The only thing that needs to be done is to configure the exports. This file assesses which directories are exported to which clients.

Listing 7.3: NFS configuration (/etc/exports)

```
# July 05 / Mathias Payer (payerm@student.ethz.ch)
# Here all clients are listed
# (all clients share the same directory)
/home/kiosks/default kiosk*.ethz.ch(ro,no_root_squash,sync)
# /home/kiosks/default \
#    kiosk01.ethz.ch(ro,no_root_squash,sync)
```

7.1.1.4 Boot parameters per client

The tftpboot directory contains a symlink to the bootloader (named pxelinux.0), a directory with all configuration files (named pxelinux.cfg) and the different Linux kernels.

If we look into the configuration directory we see a file for each kiosk hardware type. Then for each kiosk there exists a symlink from its IP address (in hex) to its hardware configuration file.

Here we include the iris-epia configuration file for the old hardware revision.

Listing 7.4: Bootloader config (/home/kiosks/tftpboot/pxelinux.cfg/iris-epia)

```
DEFAULT kiosk
```

```
LABEL kiosk
        KERNEL vmlinuz-iris-epia-2.6.11.6
        APPEND root=/dev/nfs nfsroot=/home/kiosks/default \
        ip=bootp vga=0x317
```

Listing 7.5: Bootloader config (/home/kiosks/tftpboot/pxelinux.cfg/iris-epiaII)

```
DEFAULT kiosk
LABEL kiosk
        KERNEL vmlinuz-iris-epia-2.6.11.6
        APPEND root=/dev/nfs nfsroot=/home/kiosks/default \
        ip=bootp vga=0x31B
```

Additionally a symlink from "8184714D" (this is 129.132.113.77 in hex, without dots) to "iris-epia" exists to indicate that this hardware revision will load the kernel "vmlinuz-iris-epia-2.6.11.6" and then mount the directory "/home/kiosks/default" as remote root. The second kiosk has the IP address 129.132.113.74 and has therefore a symlink from "8184714A" to "iris-epiaII".

7.1.1.5 Package list

To simplify a reinstallation of the server we provide a list of all installed packages.

adduser	bison	defoma
apache-common	bsdmainutils	dhcp
apache2	bsdutils	dictionaries-common
apache2-common	bzip2	diff
apache2-mpm-prefork	console-common	discover1
apache2-utils	console-data	discover1-data
apt	console-tools	dnsutils
apt-utils	coreutils	doc-debian
aptitude	cpio	doc-linux-text
at	cpp	dpkg
atftpd	cpp-3.3	dpkg-dev
base-config	cramfsprogs	dselect
base-files	cron	e2fslibs
base-passwd	dash	e2fsprogs
bash	dc	e3
bc	debconf	ed
bin86	debconf-i18n	eject
bind9-host	debianutils	fdutils
binutils	debootstrap	file

findutils
finger
flex
fontconfig
ftp
g++
g++-3.3
gawk
gcc
gcc-3.3
gcc-3.3-base
gdb
gettext-base
gnu-efi
gnupg
grep
groff-base
grub
gzip
hostname
hotplug
iamerican
ibritish
ifupdown
imagemagick
info
initrd-tools
initscripts
ipchains
iptables
iputils-ping
ispell
kernel-image-2.6...
kernel-image-2.6...
kernel-package
klogd
less
libacl1
libapache-mod-php4
libapache2-mod-php4
libapr0
libattr1
libblkid1
libbz2-1.0
libc6
libc6-dev
libcap1
libcomerr2
libconsole
libdb1-compat
libdb3
libdb4.2
libdb4.3
libdbd-mysql-perl
libdbi-perl
libdiscover1
libdns16
libdps1
libexpat1
libfontconfig1
libfreetype6
libgc1
libgcc1
libgcrypt11
libgd2-xpm
libgdbm3
libgnutls11
libgpg-error0
libgpmg1
libice6
libident
libidn11
libisc7
libjasper-1.701-1
libjpeg62
libkrb53
liblcms1
libldap2
liblocale-gett...
liblockfile1
liblwres1
liblzo1
libmagic1
libmagick6
libmd5-perl
libmysqlclient12
libncurses5
libncurses5-dev
libncursesw5
libnet-daemon-perl
libnewt0.51
libnfsidmap1
libnss-db
libopencdk8
libpam-modules
libpam-runtime
libpam0g
libpcap0.7
libpcre3
libplrpc-perl
libpng12-0
libpopt0
libreadline4
libreadline5
libsasl2
libsigc++-1.2-5c102
libsm6
libss2
libssl0.9.7
libstdc++5
libstdc++5-3.3-dev
libt1-5
libtasn1-0
libtasn1-2
libtext-charwidt...
libtext-iconv-perl
libtext-wrapi18n...

libtextwrap1
libtiff4
libusb-0.1-4
libuuid1
libwrap0
libx11-6
libxext6
libxml2
libxpm4
libxt6
libzzip-0-12
links
linux-kernel-hea...
locales
login
logrotate
lpr
lsof
m4
mailx
make
makedev
man-db
manpages
manpages-dev
mawk
mime-support
module-init-tools
modutils
mount
mpack
mtools
mtr-tiny
mutt
mysql-client
mysql-common
mysql-server
nano
ncurses-base
ncurses-bin
ncurses-term
net-tools
netbase
netkit-inetd
nfs-common
nfs-kernel-server
ntpdate
nvi
openssl
passwd
patch
pciutils
perl
perl-base
perl-modules
php4
php4-common
php4-gd
php4-imagick
php4-mysql
pidentd
portmap
ppp
pppconfig
pppoe
pppoeconf
procmail
procps
psmisc
python
python-newt
python2.3
rcs
reportbug
sed
sharutils
slang1
slang1a-utf8
smail
ssh
ssl-cert
strace
sysklogd
syslinux
sysv-rc
sysvinit
tar
tasksel
tcpd
tcsh
telnet
texinfo
time
traceroute
ttf-bitstream-vera
ucf
unzip
usbutils
util-linux
w3m
wamerican
wget
whiptail
whois
xfree86-common
xlibs-data
zip
zlib1g

7.1.2 Client side

Now we will show what has to be done to get the clients up and running. Acctually all these files are in subdirectories of the server, but we call it client side as these files are only needed on the kiosks and not on the server.

7.1.2.1 Mounted filesystems (/etc/fstab)

As the client has no local storage and loads all the data over the network only three different filesystems are listed there. The proc filesystem that handles the kernel runtime configuration (this is needed during startup and for the services). The sysfs filesystem, which is needed for the USB TFT touchscreens and the root filesystem that is mounted via NFS.

Listing 7.6: Filesystem table on client (/etc/fstab)

```
# fstab for client kiosks
# May 05 / Mathias Payer (payerm@student.ethz.ch)
# <fsy> <mount> <type> <options> <dump> <pass>
proc      /proc proc  defaults 0 0
/dev/nfs  /     nfs   defaults 0 1
sysfs     /sys  sysfs defaults 0 0
```

7.1.2.2 Special boot script (/etc/init.d/config_kiosk.sh)

On a standard Linux distribution like Debian, some directories need write access. One possibility would be to change all the startup scripts, but this would mean a lot of work. The easier solution is to create a ramdisk and copy all the files that need to be changed to this ramdisk. Then there is a symlink from the original location to this ramdisk and so the scripts can write to these files. After startup Mozilla Firefox also wants to write into the profile directory, so we also include our standard user's home directory into the ramdisk. As we do not care about these files, we do not mind that all changes will be lost after a reboot.

Listing 7.7: Boot script to configure remote kiosks (/etc/init.d/config_kiosk.sh)

```
# This is the config script that creates all symlinks to the
# temporary filesystem, so that logfiles can be "written"
# and temporary files created.
# July 05 / Mathias Payer (payerm@student.ethz.ch)

echo Configuring diskless operation...
```

```
# get my ip to setup special direcrtories...

MYIP=`/sbin/ifconfig eth0 | grep inet | awk '{print $2}'\
        | sed 's/^addr://g'`
echo My ip: $MYIP

# creating virtual "writable" dir with all special dirs
# in it that need rw-access.
# So we mount a temporary, volatile ram-disk that will save
# all out temporary data (and kepp thinks like home
# directory and mozilla profiles)

# mount ramfs
mount -t tmpfs tmpfs /var/virt

# temp-directory (there is also a symlink from /tmp)
mkdir /var/virt/tmp
# symlink from /var/run
mkdir /var/virt/run
# symlink from /var/log
mkdir /var/virt/log
# symlink from /var/lock
mkdir /var/virt/lock
# symlink from /home/kiosk
mkdir /var/virt/kiosk

mkdir -p /var/virt/lib/urandom
chown kiosk.kiosk /var/virt/kiosk

# setting up "home-directory" of standard-user
# we include the special directory with the mozilla-
# profile for the right screen resolution.
tar cf - -C `cat /opt/$MYIP/mozilla_profile` --exclude .. \
        .* | tar xf - -C /home/kiosk/

# linking machine-specific configuration to the central
# configuration directory
# here we will save things like XF86Config and start-URL
ln -s /opt/$MYIP /var/virt/configs
```

```
# log files for the USB-Elo driver.
mkdir -p /var/log/zx_elo
touch /var/log/zx_elo/log.txt
chmod 666 /var/log/zx_elo/log.txt

exit 0
```

7.1.2.3 XF86Config (/opt/<IP>/XF86Config-4)

These configuration files are very machine dependent. The first configuration file is for the old TFT Monitor that uses a serial interface to communicate. The second one uses USB to signal touch events on the screen. To limit the length of this document we only included the relevant parts of the configuration file, as these configuration files tend to be quite long.

Listing 7.8: Config of the GUI and TFT (/etc/X11/XF86Config-4 for kiosk01)

```
Section "InputDevice"
        Identifier      "Generic Keyboard"
        Driver          "keyboard"
        Option          "CoreKeyboard"
        Option          "XkbRules"      "xfree86"
        Option          "XkbModel"      "pc104"
        Option          "XkbLayout"     "de_CH"
        Option          "XkbOptions"    "nodeadkeys"
EndSection

Section "InputDevice"
        Identifier      "Configured Mouse"
        Driver          "mouse"
        Option          "CorePointer"
        Option          "Device" "/dev/psaux"
        Option          "Protocol" "PS/2"
        Option          "Emulate3Buttons" "true"
        Option          "ZAxisMapping" "4 5"
EndSection

Section "InputDevice"
        Identifier      "Generic Mouse"
        Driver          "mouse"
```

```
        Option          "SendCoreEvents" "true"
        Option          "Device" "/dev/input/mice"
        Option          "Protocol" "ImPS/2"
        Option          "Emulate3Buttons" "true"
        Option          "ZAxisMapping" "4 5"
EndSection

# Touchscreen configuration
# The Min/Max[X/Y] are very important!
Section "InputDevice"
        Identifier      "touchscreen"
        Driver          "elographics"
        Option          "Device"        "/dev/ttyS0"
        Option          "AlwaysCore"
        Option          "screenno"      "0"
        Option          "MinX"  "4000"
        Option          "MaxX"  "100"
        Option          "MinY"  "4000"
        Option          "MaxY"  "100"
        Option          "UntouchDelay"  "3"
        Option          "ReportDelay"   "1"
EndSection

Section "Device"
        Identifier      "Generic Video Card"
        Driver          "vesa"
EndSection

Section "Monitor"
        Identifier      "Generic Monitor"
        HorizSync       28-50
        VertRefresh     43-75
        Option          "DPMS"
EndSection

Section "Screen"
        Identifier      "Default Screen"
        Device          "Generic Video Card"
        Monitor         "Generic Monitor"
```

```
        DefaultDepth     24
        SubSection "Display"
                Depth           24
                Modes           "1280x1024"
        EndSubSection
EndSection

Section "ServerLayout"
        Identifier      "Default Layout"
        Screen          "Default Screen"
        InputDevice     "Generic Keyboard"
        InputDevice     "Configured Mouse"
        InputDevice     "Generic Mouse"
        # use touchscreen as input device
        InputDevice     "touchscreen"
EndSection

# To keep the tft from going into
# economy-mode we use some special
# flags
Section "ServerFlags"
        Option  "BlankTime"     "0"
        Option  "StandbyTime"   "0"
        Option  "SuspendTime"   "0"
        Option  "OffTime"       "0"
EndSection
```

The second TFT screen uses an USB interface. Some drivers are supplied by ELO that should work with XFree86. The problem is that these drivers are faulty (the x-axis is interpreted as y-axis and the y-axis is lost). ELO has other drivers for registered users on their web page. The problem with these drivers is that they are only for a very old kernel version (2.4) and not for newer systems.

Somebody corrected the XFree OpenSource drivers[17] from ELO and released these drivers on his website. We used this driver and were then able to work with the USB interface.

Listing 7.9: Config of the GUI and TFT (/etc/X11/XF86Config-4 for kiosk02)

```
Section "InputDevice"
        Identifier      "Generic Keyboard"
        Driver          "keyboard"
```

```
        Option          "CoreKeyboard"
        Option          "XkbRules"      "xfree86"
        Option          "XkbModel"      "pc104"
        Option          "XkbLayout"     "de_CH"
        Option          "XkbOptions"    "nodeadkeys"
EndSection

Section "InputDevice"
        Identifier      "Configured Mouse"
        Driver          "mouse"
        Option          "CorePointer"
        Option          "Device"                "/dev/psaux"
        Option          "Protocol"              "ImPS/2"
        Option          "Emulate3Buttons"       "true"
        Option          "ZAxisMapping"          "4 5"
EndSection

Section "InputDevice"
        Identifier      "Generic Mouse"
        Driver          "mouse"
        Option          "SendCoreEvents"        "true"
        Option          "Device"        "/dev/input/mice"
        Option          "Protocol"              "ImPS/2"
        Option          "Emulate3Buttons"       "true"
        Option          "ZAxisMapping"          "4 5"
EndSection

Section "InputDevice"
        Identifier      "touchscreen"
        Driver          "elousb"
        Option          "Device"        "/dev/input/event1"
        Option          "AlwaysCore"
#       Option          "screenno"      "0"
#       Option          "rotate"        "CW"
#       Option          "devicename"    "IntelliTouch 2500U"
        Option          "MinX"  "4000"
        Option          "MaxX"  "10"
        Option          "MaxY"  "4095"
        Option          "MinY"  "98"
```

```
        Option          "InputFashion"  "Touchpanel"
        Option          "ReportingMode" "scaled"
        Option          "ScreenNumber"  "1"
        Option          "SwapAxes"      "0"
        Option          "UntouchDelay"  "3"
        Option          "AlwaysCore"    "On"
        Option          "ReportDelay"   "1"
        Option          "debuglevel"    "5"
EndSection

Section "Device"
        Identifier      "Generic Video Card"
        Driver          "vesa"
EndSection

Section "Monitor"
        Identifier      "Generic Monitor"
#       HorizSync       28-50
#       VertRefresh     43-80
        Option          "DPMS"
EndSection

Section "Screen"
        Identifier      "Default Screen"
        Device          "Generic Video Card"
        Monitor         "Generic Monitor"
        DefaultDepth    24
        SubSection "Display"
                Depth           1
                Modes           "1280x1024"
        EndSubSection
        SubSection "Display"
                Depth           4
                Modes           "1280x1024"
        EndSubSection
        SubSection "Display"
                Depth           8
                Modes           "1280x1024"
        EndSubSection
```

```
        SubSection "Display"
                Depth           15
                Modes           "1280x1024"
        EndSubSection
        SubSection "Display"
                Depth           16
                Modes           "1280x1024"
        EndSubSection
        SubSection "Display"
                Depth           24
                Modes           "1280x1024"
        EndSubSection
EndSection

Section "ServerLayout"
        Identifier      "Default Layout"
        Screen          "Default Screen"
        InputDevice     "Generic Keyboard"
        InputDevice     "Configured Mouse"
        InputDevice     "Generic Mouse" "SendCoreEvents"
        InputDevice     "touchscreen" "SendCoreEvents"
EndSection

# To keep the tft from going into
# economy-mode we use some special
# flags
Section "ServerFlags"
        Option  "BlankTime"     "0"
        Option  "StandbyTime"   "0"
        Option  "SuspendTime"   "0"
        Option  "OffTime"       "0"
EndSection
```

7.1.2.4 Autologin and xinit

After system startup has completed the user “kiosk” is automtically logged into the first terminal and his starup scripts and profiles are executed. To autologin the kiosk user the file “/etc/inittab” was altered.

Listing 7.10: Autologin for the kiosk user (/etc/inittab)

```
# To use autologin we need to change the following line
#1:2345:respawn:/sbin/getty 38400 tty1
# into
1:12345:respawn:/sbin/mingetty --noclear --autologin \
    kiosk tty1
```

This leads as to a console with the user "kiosk" logged in. Now we changed the bash (the standard shell) profile for this user as follows:

Listing 7.11: Bash profile for the kiosk user (/.bash_profile)

```
# append these lines at the end of the file:
# start X (GUI) and execute only mozilla-firefox
# w/out any window-environment
# use the content of the file 'url' as the first
# page that is to be displayed
xinit /usr/bin/mozilla-firefox `cat url`
# if something goes wrong and mozilla terminates
# we are automatically logged out
logout
# as we are logged out, the terminal tries to
# respawn and kiosk is logged in again and
# mozilla will restart
```

7.1.2.5 Kernel configuration

The configuration of the kernels differs from kiosk to kiosk. One has to include the necessary drivers for each hardware revision.

We provide here only the important options that are needed to provide root over nfs and networkin functionality.

CONFIG_ROOT_NFS With this option the client tries to mount the given nfsroot.

CONFIG_SOCKET, CONFIG_FILTER These options need to be enabled to use DHCP configuration.

CONFIG_IP_PNP, ...PNP_DHCP, ...PNP_BOOTP With these options the kernel is able to autoconfigure itself (without the need of external programs like dhclient and others).

7.1.2.6 Package list

To easy a reinstallation of the server we provide a list of all installed packages.

```
adduser
apt
apt-utils
aptitude
base-files
base-passwd
bash
bc
bind9-host
binutils
bsdmainutils
bsdutils
bzip2
console-common
console-data
console-tools
coreutils
cpp
cpp-3.3
cron
dash
dc
debconf
debconf-i18n
debianutils
deborphan
defoma
dialog
dictionaries-common
diff
discover1
discover1-data
dnsutils
dpkg
dselect
e2fslibs
e2fsprogs
e3
ed
fbset
file
findutils
fontconfig
gcc-3.3-base
gettext-base
gnu-efi
gnupg
grep
groff-base
gzip
hostname
hotplug
ifupdown
initscripts
iptables
iputils-ping
kernel-headers-2...
kernel-image-2.6...
klogd
less
libacl1
libatk1.0-0
libattr1
libblkid1
libbz2-1.0
libc6
libcap1
libcomerr2
libconsole
libdb1-compat
libdb3
libdb4.2
libdiscover1
libdns16
libdps1
libexpat1
libfontconfig1
libfreetype6
libgcc1
libgcrypt11
libgdbm3
libglib2.0-0
libgnutls11
libgpg-error0
libgtk2.0-0
libgtk2.0-bin
libgtk2.0-common
libice6
libident
libidl0
libisc7
libjpeg62
libkrb53
libldap2
liblocale-gett...
liblwres1
liblzo1
libmagic1
libncurses5
libncursesw5
libnewt0.51
libopencdk8
libpam-modules
libpam-runtime
libpam0g
libpango1.0-0
libpango1.0-common
libpng12-0
```

```
libpopt0                  lsof                      rxvt
libreadline4              m4                        sed
libreadline5              makedev                   sharutils
libsasl2                  man-db                    slang1
libsigc++-1.2-5c102       manpages                  slang1a-utf8
libsm6                    mawk                      smail
libss2                    mime-support              ssh
libssl0.9.7               mingetty                  strace
libstdc++5                module-init-tools         sysklogd
libtasn1-2                modutils                  sysv-rc
libtext-charwidt...       mount                     sysvinit
libtext-iconv-perl        mozilla-firefox           tar
libtext-wrapi18n...       mpack                     tcpd
libtextwrap1              nano                      tcsh
libtiff4                  ncurses-base              telnet
libusb-0.1-4              ncurses-bin               time
libuuid1                  ncurses-term              traceroute
libwrap0                  net-tools                 ttf-bitstream-vera
libx11-6                  netbase                   ucf
libxaw7                   netkit-inetd              udev
libxcursor1               nfs-common                usbutils
libxext6                  ntpdate                   util-linux
libxft1                   passwd                    wget
libxft2                   pciutils                  xbase-clients
libxi6                    perl                      xfonts-100dpi
libxmu6                   perl-base                 xfonts-75dpi
libxmuu1                  perl-modules              xfonts-base
libxp6                    pidentd                   xfonts-scalable
libxpm4                   portmap                   xfree86-common
libxrandr2                procmail                  xlibmesa-gl
libxrender1               procps                    xlibmesa-glu
libxt6                    psmisc                    xlibs
libxtrap6                 python                    xlibs-data
libxtst6                  python-newt               xserver-common
libxv1                    python2.3                 xserver-xfree86
locales                   rcs                       xutils
login                     reportbug                 zlib1g
```

7.1.3 CMS Configuration

The configuration of the chosen CMS (typo3 in our case) is quite easy. First of all we need a database, so we start mysql and create a new database named typo3 with "create database typo3". Then we must add a new user that is able to update this database with "GRANT ALL PRIVILEGES ON typo3.* TO 'typo3'@'localhost' IDENTIFIED BY 'some_pass' WITH GRANT OPTION;".

Then we install the typo3-tarball into our webserver directory (we must make sure, that all needed software like PHP, MySQLP, Apache2 and so on is installed!) and execute the installation procedure.

This process of installing the typo3 database is explained in a lot of detail in this howto: http://typo3.org/documentation/document-library/doc_inst_upgr/ so we don't include these steps in this document.

7.2 Used Hardware

Now we will present the used hardware. One of the design goals of the multimedia kiosk is to use as much standard "off-the-shelf" hardware as possible. This has the advantages that the kiosks are cheap and that there drivers exist for all hardware parts.

7.2.1 Kiosk-Hardware

The kiosks are basicially standard Intel compatible computers. They use mainboards with a special small form factor (called mini-ITX) and low power consuming (fanless) processors. The mainboards and processors are factured by VIA Computers LTD. One of the few features that is needed is direct BIOS support for boot over LAN, but nearly all of these mini-ITX boards from VIA include this support.

The old kiosk client uses a VIA Epia MII mainboard with a fanless 600 Mhz processor. This mainboard also has a lot of additional (but in our kiosk unused) features like a CF card slot and a PCMCIA slot.

The new kiosk uses a VIA Epia ML mainboard with a fanless 1 GHz processor. This new mainboard should provide much better hardware stability and the faster processor makes a difference when bigger webpages are displayed.

7.2.1.1 Touchscreens

The touchscreens we use are from ELO Touch Systems[16]. One of the two monitors was already used in an earlier project. This touchscreen is controlled using a serial interface. It is not as exact as the new one but it is enough to be used as a test environment.

The new monitor is an ELO ET1947L. This one is delivered without case and can be mounted on the wall. It uses an USB interface to the touchscreen and is used in the new kiosk client.

7.2.1.2 Server Hardware

The server is not very resource hungry. In the lab we used an "off-the-shelf" DELL computer. The only important things are that the server has enough memory (about 512 to 1024 should be sufficient) and a RAID disk. As all data is saved centrally on the server a disk failure there would lead to a lot of problems as all data and all kiosk configurations would be lost.

Bibliography

[1] **Institute of Robotics and Intelligent Systems:**
http://www.iris.ethz.ch

[2] **Master Kiosk-Server:**
http://kiosk-server.ethz.ch

[3] **Linux Documentation Project:**
http://www.tldp.org/

[4] **Root-over-NFS Howto:**
http://www.tldp.org/HOWTO/Diskless-root-NFS-HOWTO.html
http://www.tldp.org/HOWTO/NFS-Root.html
http://www.tldp.org/HOWTO/NFS-Root-Client-mini-HOWTO/
http://www.tldp.org/HOWTO/Diskless-root-NFS-other-HOWTO.html

[5] **Linux-PXE Boot Howto:**
http://www.tldp.org/HOWTO/Remote-Boot.html
http://www.kegel.com/linux/pxe.html
http://syslinux.zytor.com/pxe.php

[6] **Wake on LAN:**
http://gsd.di.uminho.pt/jpo/software/wakeonlan/mini-howto/

[7] **EPIA Via Homepage:**
http://www.viac3.de/

[8] **Mainboards used as kiosks:**
Old kiosk:
http://www.viavpsd.com/product/epia_MII_spec.jsp?motherboardId=202
New kiosk:
http://www.viavpsd.com/product/epia_ms_spec.jsp?motherboardId=281
http://www.mini-itx.com/store/?c=2#p1624

[9] **Mini-ITX resources and shop:**
http://www.mini-itx.com/

[10] **Debian - Linux distribution:**
http://www.debian.org/

[11] **Linux administration:**
http://www.tldp.org/LDP/sag/html/index.html
http://www.tldp.org/LDP/gs/gs.html
http://www.debian.org/doc/
http://www.debian.org/doc/user-manuals#install

[12] **Typo3 - CMS on master kiosk:**
http://www.typo3.com
Documentation:
http://typo3.org/1421.0.html
New user quickstart tutorial:
http://typo3.org/documentation/document-library/doc_tut_quickstart/

[13] **Kerel Documentation:**
Have a look at the file Documentation/nfsroot.txt (via the root of the linux kernel source)

[14] **Diskless Linux boot PXE howto:**
http://www.intra2net.com/opensource/diskless-howto/howto.html

[15] **Adapted kiosk-mozilla:**
http://kiosk.mozdev.org/

[16] **ELO Touch Systems:**
http://www.elotouch.com/

[17] **Driver for ELO USB Touchscreens:**
The original ELO Driver for XFree is faulty and does not work. The following driver has the corrected sources.
http://www.softcoded.net/eduard/elousb.html

Glossary

APM ..... **A**dvanced **P**ower **Management** is used to cut most of the mainboard from power after a shutdown and to control fans and power usage.

BIOS ..... **B**asic **I**nput **O**utput **S**ystem. When a computer is started then the memory is completely blank. The BIOS is the first part that is loaded. The BIOS checks then for available hardware and starts a POST. After all checks are passed the BIOS tries to load the bootloader (depending on the BIOS configuration either from harddisc, CDROM, network, ...).

bootloader ..... After a computer has completed the POST it tries to load an OS. The problem is, that the OS is too big to be loaded directly by the BIOS. So the startup is divided into two stages. First the BIOS loads the bootloader, then the bootloader loads the actual kernel. So the bootloader is a small program (loaded by the BIOS) that loads the OS (kernel) and then passes control to the kernel (thereby starting the OS.)

broadcast ..... The sender addresses multiple machines.

CMS ..... **C**ontent **M**anagement **S**ystem this is a way to organize webpages over an easy to use webinterface.

DHCP ..... **D**ynamic **H**ost **C**onfiguration **P**rotocol. A special protocol to configure IP-Networking without any client configuration.

IP ..... **I**nternet **P**rotocol (RFC 791), a protocol that defines how two devices can communicate with each other over the internet.

kernel ..... Every unix like system has a system kernel. The kernel is started when the computer boots and controls all hardware I/O (input/output) and all devices. It also separates the different users and running processes.

MAC address .... **M**edia **A**ccess **C**ontrol address, a globally unique address for a network card. These addresses are associated by the IEEE.

NFS ..... **N**etwork **F**ile **S**ystem. This system was developed by Sun Microsystems to export files and directories over a network.

OS ..... **O**perating **S**ystem. Every computer needs an OS to run. On unix systems the kernel is the base of the operating system. Other utilities are onion-like layered around the kernel. All tools together form the OS.

POST ..... **P**ower **O**n **S**elf **T**est. Everytime a computer starts it checks if vital components are working (like memory, cpu, graphic card, network adapter, ...). If a component fails then a special beep-code informs about the failure.

RFC ..... **R**equest **F**or **C**omments, a document series that handles the definition of protocols used in the internet. Every proposal gets a unique number and can then be discussed. For an overview about existing pages see http://www.ietf.org/rfc.html.

TCP ..... **T**ransmission **C**ontrol **P**rotocol (RFC 793) defines a protocol that builds upon IP and offers a continous data-stream with error correction, data serialisation (all packets arrive in order) and the concepts of connections.

TFTP ..... **T**rivial **F**ile **T**ransfer **P**rotocol. An easy way to transfer files. The protocol was kept as simple as possible to easy the transport of bootup images and configuration files to diskless clients.

UDP ..... **U**ser **D**atagram **P**rotocol (RFC 768), a protocol that builds upon IP and offers a connectionless possibility to send packets to a specific host.

unicast ..... The sender addresses exactly one machine.

URL ..... **U**niform **R**essource **L**ocation, a way to specify protocol and location of documents (e.g. http://www.iris.ethz.ch/index.html specifies that the document index.html is available via http in the document root of the www.iris.ethz.ch server).

Listings

Chapter 8

Statutory declaration

I Mathias Payer do solemnly and sincerly declare and affirm that:

1. this thesis was written by myself
2. only the documents in the bibliography were used
3. all references to external documents are included

and I make this solemn declaration, as to the aforesaid, according to the laws in this behalf made, and subject to the punishment by law provided for any wilfully false statement in any such declaration.