Design and implementation of recording functionality for an IP-based set-top box

by

Aner Gusic

LITH-ISY-EX--05/3665--SE

Linköping, 2005
Design and implementation of recording functionality for an IP-based set-top box

Examensarbete utfört i datorteknik
vid Linköpings tekniska högskola
av

Aner Gusic

LITH-ISY-EX--05/3665--SE

Supervisor: Henrik Carlsson
R & D
at Kreatel Communications AB

Examiner: Ph.D. Dake Liu
Dept. of Computer and Information Science
at Linköpings universitet

Linköping, 2005-03-16
This thesis covers the design and implementation of recording functionality for a set-top box in a home network. An initial investigation is done and possibilities for extending the system to support specific features are presented.

Digital TV is becoming more common each day, and soon it will be more widely used than today's analogue standard. At the same time the need for recording TV shows remains the same or is increasing, which is shown by the number of PVR solutions popping up on the market.

The goal of this thesis work was to investigate the possibilities for extending an existing set-top box to support common PVR features and, if possible, to implement a prototype. This was supposed to be done in a home network environment with the set-top box as the digital media center.

A satisfying solution, covering basic recording functionality is defined and implemented. The solution includes recording to a USB hard drive and to a PC on the local network. On top of this, a graphical user interface is built and some simple benchmarks show the performance of the set-top box with the new functionality.
Abstract

This theses covers the design and implementation of recording functionality for a set-top box in a home network. An initial investigation is done and possibilities for extending the system to support specific features are presented.

Digital TV is becoming more common each day, and soon it will be more widely used than today’s analogue standard. At the same time the need for recording TV shows remains the same or is increasing, which is shown by the number of PVR solutions popping up on the market.

The goal of this thesis work was to investigate the possibilities for extending an existing set-top box to support common PVR features and, if possible, to implement a prototype. This was supposed to be done in a home network environment with the set-top box as the digital media center.

A satisfying solution, covering basic recording functionality is defined and implemented. The solution includes recording to a USB hard drive and to a PC on the local network. On top of this, a graphical user interface is built and some simple benchmarks show the performance of the set-top box with the new functionality.

Keywords: PVR, DVR, PDR, set-top box, streaming, recording, time-shift, digital media, IPTV
Acknowledgements

I would like to thank my supervisor Henrik Carlsson for giving me the opportunity to do an interesting thesis work and for the guidance while doing it. I also received a lot of help from developers, especially Daniel and Oscar, without whom this thesis would not be the same. Thanks for pointing out the worst pitfalls, before I walked into them.

Then we have the people involved in the written part of the thesis work, my opponent Manne Anliot and a friend of mine Ulrik Haugen. It is fascinating how many times you can read a paper and miss simple misspellings or not notice lack of entire sections. Thanks both for making my thesis better!

I also feel gratitude to my examiner, Dake Liu, for his advice and for giving me free hands while working with this thesis. Unfortunately, this doesn’t always happen as a matter of course.

Also many thanks to everyone making the sometimes long office hours, shorter. This includes Per Engström, my “office mate” and everybody besotted with table tennis during lunch breaks!

Finally, I would like to express my gratitude towards the open source community. The set-top box, server software I used and my workstation all run free, open source software. \LaTeX\ was used to write this document. All those things shouldn’t be taken for granted. Thanks guys!
Contents

List of Figures ................................................................. 1

1 Introduction ....................................................................... 3
  1.1 Glossary ......................................................................... 3
  1.2 Background ...................................................................... 3
  1.3 Objectives ......................................................................... 4
    1.3.1 Original objectives ................................................... 4
    1.3.2 Latter objectives ....................................................... 5
  1.4 Limitations ........................................................................ 5
  1.5 Outline ............................................................................. 5
  1.6 Reading instructions ........................................................ 6
    1.6.1 Document style .......................................................... 7

2 Existing Solutions Survey .................................................. 9
  2.1 Background ........................................................................ 9
  2.2 TV-Anytime ........................................................................ 9
  2.3 Existing products ............................................................. 10
    2.3.1 MythTV and Freevo .................................................. 10
    2.3.2 TiVo .......................................................................... 12
    2.3.3 Force ......................................................................... 12
      Instant and scheduled recording ...................................... 12
      Time-shift ....................................................................... 13
  2.4 Storage ............................................................................ 13
3 Current system state and requirements 15
3.1 Overall system description .................................. 15
  3.1.1 System start-up ........................................... 15
  3.1.2 External connectors ...................................... 16
3.2 System hardware .............................................. 16
3.3 Software ....................................................... 17
  3.3.1 User applications ......................................... 17
  3.3.2 Services .................................................... 18
  3.3.3 The operating system and the hardware ............... 18
  3.3.4 Important modules ........................................ 18
  3.3.5 Streamer .................................................. 20
  3.3.6 Media manager ............................................ 20
  3.3.7 User application ......................................... 20
  3.3.8 URI loader ................................................ 21
3.4 Requirements .................................................. 21

4 System architecture 23
4.1 Extending the streamer ........................................ 23
  4.1.1 Extended streamer models ............................... 23
  4.1.2 Extending with elements capable of writing .......... 24
  4.1.3 Properties of write-elements ........................... 25
  4.1.4 Extending with elements capable of writing and reading 26
  4.1.5 Conclusion ............................................... 27
  4.1.6 Content protection ....................................... 27
          Encryption model ......................................... 27
          Algorithm .................................................. 27
          Encryption and decryption key .......................... 28
4.2 Extending the system architecture .......................... 28
  4.2.1 PVR demo ................................................ 28
  4.2.2 TV application .......................................... 30
  4.2.3 VoD application .......................................... 30
  4.2.4 Interfaces ................................................ 30
4.3 Implications to the chosen system solution ................ 31
  4.3.1 USB disk ............................................... 31
  4.3.2 Network storage ......................................... 31
## 5 Design and Implementation

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Modules</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>Design issues and decisions</td>
<td>35</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Protocol between the set-top box and the PC</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Existing protocols</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>HTTP</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>RTSP</td>
<td>37</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Protocol choice</td>
<td>37</td>
</tr>
<tr>
<td>5.3</td>
<td>Remote file system protocol</td>
<td>38</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Commands and responses</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>HELLO</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>VERSION</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>OPEN</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>CLOSE</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>PWD</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>WRITE</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>QUIT</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>READ</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Common responses</td>
<td>42</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Connecting to an RFS server</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Initial connection</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Reading and writing files</td>
<td>43</td>
</tr>
<tr>
<td>5.3.3</td>
<td>A short example</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>RFS server</td>
<td>45</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Server loop</td>
<td>45</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Data parsers</td>
<td>46</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Command parsers</td>
<td>47</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Data structures</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Session-set</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Command table</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Response table</td>
<td>49</td>
</tr>
<tr>
<td>5.4.5</td>
<td>An example</td>
<td>49</td>
</tr>
<tr>
<td>5.5</td>
<td>Diskwriter element</td>
<td>49</td>
</tr>
<tr>
<td>5.5.1</td>
<td>TDiskWriterElement</td>
<td>52</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>USB disk</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>RFS</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>5.5.2 TRfsClient</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>5.5.3 IEvent and IEventCallback</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>5.5.4 IEventManager</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>5.5.5 TResponse and IResponseCallback</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>5.5.6 Content protection</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Transport stream</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Encryption of packets</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Decryption of packets</td>
<td>56</td>
<td></td>
</tr>
</tbody>
</table>

| 5.6 PVR demo                | 56   |
| 5.6.1 Features              | 56   |
| 5.6.2 GUI                   | 57   |
| Sub-menu: status            | 57   |
| Sub-menu: view recordings   | 57   |
| Sub-menu: schedule recordings | 57   |

| 6 Discussion                | 59   |
| 6.1 From requirements to a solution | 59   |
| 6.1.1 Requirements         | 59   |
| 6.1.2 The system           | 60   |
| 6.2 Benchmarks             | 60   |
| 6.2.1 USB                  | 60   |
| 6.2.2 Encryption and CPU usage | 61   |
| The encrypted stream       | 62   |
| 6.3 Problems with the current solution | 62   |
| 6.3.1 USB interface        | 62   |
| 6.3.2 File I/O             | 63   |
| Deleting files             | 63   |
| 6.3.3 Recording awareness  | 63   |

| 7 Future work               | 65   |
| 7.1 Blocked disk access     | 65   |
| 7.1.1 Modifying the kernel  | 65   |
| 7.1.2 File writing service  | 65   |
| 7.2 Alternative streamer solution | 66   |
CONTENTS

7.2.1 Modifications to steamer ........................................... 66
7.3 Remote storage access protocols ...................................... 67
  7.3.1 HTTP and UPnP .................................................. 68
  7.3.2 RTSP .............................................................. 68
  7.3.3 RFS ............................................................... 68
7.4 Recording awareness ..................................................... 69

A Glossary ........................................................................ 70

Bibliography ..................................................................... 74
List of Figures

3.1 The set-top box model used for this thesis work ........ 16
3.2 Architectural overview of the set-top box software .... 19
3.3 Important system software components ................. 19

4.1 Streamer model capable of writing to a file or a LAN .... 24
4.2 Streamer model capable of writing to undefined storage .. 25
4.3 Streamer model capable of both reading and writing ..... 26
4.4 Updated system software overview ...................... 29

5.1 Conceptual view of a recording capable home system ... 34
5.2 Interaction diagram for RFS client and server ........ 43
5.3 RFS server architectural view .......................... 45
5.4 Parsing of data in the RFS server ........................ 46
5.5 RFS server: runtime flow chart ......................... 50
5.6 UML diagram for the diskwriter element ................ 51
5.7 State machine for the TDiskWriterElement ............. 53
5.8 Transport stream ........................................ 55

7.1 Future streamer architecture .............................. 67
Chapter 1

Introduction

This chapter will present the background and the needs that led to this thesis work. Subsequently, the objectives are described as well as the limitations for this work. The document is also described. First the document outline, then reading instructions which describe both the chapter content and the text styles used later on.

1.1 Glossary

Many acronyms and special expressions are used in this document. It is recommended to take a look at appendix A if the need arises, or before reading this document.

1.2 Background

The company interested in this thesis work makes IP-based set-top boxes and several models exist, for example a pure IP-based set-top box (requires operator with streaming services) and a hybrid IP/DVB-T box for IP/terrestrial digital TV. However, no model features recording and time-shift capabilities which are becoming more and more common in digital TV solutions.
The set-top box is mainly constructed for reception of video and audio content and rendering/playback in real time. Adding recording and/or time-shift features would mean choosing a software solution, updating the software platform and writing additional software needed to perform desirable functions.

1.3 Objectives

The original objectives were not very stringent, but instead consisted of general ideas and thoughts about how a PVR should work. Still, they served their purpose until more specific objectives crystallised as the project matured.

I feel that there is an important distinction between these two, each with their own values. That is the reason why these two sets of objectives are presented separately. The “latter objectives” will be better understood if read after the introductory chapters of this thesis.

1.3.1 Original objectives

- Investigate existing hardware and software solutions and choose a fitting set of features for this project.

- Find out how hard it is to modify the set-top box to support basic recording functionality.

- Implement a basic PVR solution for our set-top box. This might include one or several of the possible software and hardware solutions. This solution should allow performance testing in order to gather data about the set-top box while recording is in progress.

- If there is time, take a look at content protection, i.e. encrypting of permanently stored data. Look into which effects encrypting data has on the set-top box performance.
1.3.2 Latter objectives

- Investigate if the USB1 interface is enough to support recording and describe in which way it is restrictive.

- Extend the streamer\(^1\) to support recording over the local network, with everything that this implies, and test this solution.

- Find out how content protection, i.e. encryption, affects the CPU usage of the streamer.

- Write software that is sufficient for simple demonstrations of the PVR functions, both for possible customers and for the presentation of the thesis.

1.4 Limitations

This project is a master thesis and thus limited in time to approximately 20 weeks of work. Objectives might be altered (extended or reduced) in order to fit the time plan.

Doing a thesis work with a commercial product has both its positive and its negative sides. One of the negative sides is not being able to write freely about everything involving the thesis work. Due to the NDA\(^2\), many implementation details are left out of the report.

1.5 Outline

This chapter is an introduction, both to the thesis subject and to the report. In chapter 2, existing solutions and solution specific features are presented.

Chapter 3 describes the system, both the hardware and the software aspects. Interesting parts of the software are described in more detail. The chapter ends with requirements for this thesis work.

\(^1\)Software component which plays audio/video streams and has a central role in this thesis.

\(^2\)Non-disclosure agreement.
Next two chapters, 4 and 5, describe architectural and design decisions, the architectural choices that were made and the design of individual software modules. These two chapters describe the changes to the set-top box which were made to implement the recording functionality.

Chapter 6 is about results and reflections on the thesis work. The results are complemented with some benchmarks on CPU and bus usage. Finally, shortcomings of the selected solution and/or the implementation are discussed. This discussion is continued in chapter 7, which is about further development of the system. Here, alternative solutions are discussed and advice is given to those responsible for the continuing work on this project.

1.6 Reading instructions

Those who read this document will probably be interested in different aspects of it. This section aims to help the reader who doesn’t want to read the entire document, i.e. someone in need of specific information. The reader who is interested in the entire document can skip to section 1.6.1.

Those interested in an overview of the document and description of the thesis should read the abstract and chapter 1. If there is further interest in the PVR technology, chapter 2 might also be interesting.

The reader with interest in the architecture of the set-top box previous to this project, should find chapter 3 interesting.

If the main interest is in what changes were made to the set-top box during this thesis work, chapters 4 and 5 (architecture and design) should be read. The reader unfamiliar with the current system should, prior to these chapters, read chapter 3.

Chapters 6 and 7 describe what has been done and how well the chosen changes to the set-top box work. The reader interested only in results and general discussion would do well to read these chapters. If the reader has little, or no knowledge about the modified set-top box, chapter 1, but mostly sections 1.2 and 1.3, should be read first.
1.6.1 Document style

A couple of different text styles are used throughout this document. This section explains what it means when text is formatted in a special way.

<table>
<thead>
<tr>
<th>Example text</th>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sometimes, text needs to be <em>emphasised</em>.</td>
<td>Expanded abbreviations, important or special expressions.</td>
</tr>
<tr>
<td>Now we refer to the C function <code>printf</code> and protocol keywords such as GET or QUIT.</td>
<td>Code, module names, protocol keywords or similar.</td>
</tr>
<tr>
<td><a href="http://liu.se/">http://liu.se/</a> is a valid URL.</td>
<td>URLs and email addresses.</td>
</tr>
<tr>
<td>This is small text.</td>
<td>Used as figure and table text.</td>
</tr>
</tbody>
</table>
Chapter 2

Existing Solutions Survey

2.1 Background

PVRs are becoming quite common today and some attempts at standardisation have been done in the area. Main organisation for standards in this area is the tv-anytime forum, http://www.tv-anytime.org/.

2.2 TV-Anytime

TV-Anytime members have produced a number of requirement specifications, and other documents. However, most of the documents target the entire “content provider”—PVR scheme and are therefore to high-level and out of the scope for this project. Another, well debated subject is business models, broadcast models and the way they interleave with each other [1].

There are however more detailed specifications, but those target ECG and metadata handling and are not applicable to this project.
2.3 Existing products

Right now we are in the middle of a PVR boom, with new products presented more and more often. A look at these products can be of benefit for both “feature ideas” and to see what customers expect of a PVR today. As an example, we can assume that a customer won’t accept a solution without a functionality of a common VCR.

There are many PVR solutions today and some strategic selections have been made since there is not enough time to investigate all of them. (Nor any real advantage to it, since these products are quite alike.)

2.3.1 MythTV and Freevo

MythTV is free software, which can be installed on any PC running GNU Linux and equipped with adequate PVR-hardware [2]. The same applies to Freevo which I will not mention any more since its functionality is almost identical to that of MythTV [3].

MythTV is very competent software, extending far beyond a simple PVR device. Main areas MythTV can be used for are:

- **TV**
  Watch TV, use PVR functionality, view ECG information and information about programs from the Internet.

- **Music**
  Play, rip, categorise, select music. Create play-lists and more.

- **Games**
  Front-end for console emulators.

- **Images**
  View, categorise and store digital pictures.

- **Videos**
  This is a generic video player for files not created by MythTV. It also includes a dvd player and support for retrieving film information from Internet, more precisely from [http://imdb.com/](http://imdb.com/).
• Weather
  Weather module, get weather forecasts from the Internet.

  I will only go into some details for the first area of usage, i.e. “TV”. Other areas are important for a generic PVR solution, but are far out of scope for this thesis work.

  MythTV supports many PVR features, the most interesting ones described below:

• Record
  – a time interval
  – this program
  – this time-slot every day/week
  – this program any time, this/any channel

  Selection of a program to record can be done in many ways, most of them using information from the ECG.

• Manage recordings
  Delete, rename, move the recordings. An interesting feature is that deletion of files can be scheduled. This can be done on program basis, i.e. the user can choose to keep last n episodes of a show and have other deleted automatically.

• Manage conflicting recordings
  MythTV warns and gives the user the possibility to choose which show to record in case of conflicts in the recording schedule.

• Web interface to allow a user to schedule recordings remotely.

• Video editing
  Allows user to cut out parts from recorded shows, e.g. removing commercials.

• Pause live show (time-shift)
  Pause live show and continue watching later. OSD\(^1\) shows information about the length of the time-shift and free/used storage space.

---

\(^1\)“On screen display” – expression for displaying overlay information on the TV screen, e.g. a VCR displaying the play symbol when user presses the play button.
Most of these features are *high-level features* and won’t be implemented during this project. However, they should be considered during design phase of the project. Of immediate interest are features “record” and “time-shift”.

### 2.3.2 TiVo

TiVo was one of the first PVR solutions presented as well one of the most competent ones. Here I will focus on extra features TiVo provides compared to the MythTV. (However, not all MythTV features are supported by TiVo. [4])

- Support for multiple TiVo boxes.
  - Get status for all boxes in the network.
  - Transfer recordings between boxes.
  - Distributed time-shift, i.e. pause live show on one box and resume watching on another box in the network.

Interaction between boxes is far outside the scope of this project and even though it gives possibility for some interesting features, it will not be considered at all in this project.

### 2.3.3 Force

Another PVR available today is a box from Force Electronics. Due to the fact that I have access to a box made by Force Electronics it was evaluated more extensively. Here I will comment on functionality relevant to this project. This applies to a Force (series 5) box.

**Instant and scheduled recording**

Recording can be started almost instantaneously by pressing the record button or by scheduling a time-slot to record. Unfortunately, it didn’t seem to be possible to use ECG to schedule recordings, but I can not be certain since I never succeeded in downloading the ECG information.
**Time-shift**

Pros with the Force box:

- When *pause* is pressed the picture freezes instantly.
- Time-shift data can be saved to a file.
- Instant record behaves as pause (can be used as time-shift to rewind/fast forward).
- Good transition to live stream when fast-forwarding reaches end of file.
- Possibility to *rew/ff* five minutes by pressing a button.

Cons with the Force box:

- Pressing *play* after the stream has been paused doesn’t start playing from the frozen frame. Several seconds of the program are lost.
- Ending time-shift with *stop* looses several seconds (5–8 seconds) of the program.
- Pressing *record* will not start recording until after a couple of seconds.
- No information about the length of time-shift / free storage while in time-shift mode.

**2.4 Storage**

There are several possible solutions that can be used for storing recorded data and some of these are discussed in section 3.2. However, all PVRs that were looked into in this survey use a built in hard drive for storing content.
2.4. Storage
Chapter 3

Current system state and requirements

This chapter describes the system in its present state. Some software modules of special interest for this thesis work are described in more detail. At the end of the chapter, project requirements are presented.

3.1 Overall system description

From a user perspective, the set-top box looks like a common consumer electronics appliance. It is controlled via a remote control or a wireless keyboard. The set-top box and the remote control are shown in figure 3.1.

3.1.1 System start-up

The box does not contain a hard drive and the only permanent storage available is a small flash memory. This memory is used for storing the user preferences and for basic start up of the box.

After this, the set-top box software is downloaded from the tv-provider and stored in the volatile memory. It remains there until an upgrade is
initiated by the content provider or until some failure causes the system to reboot.

3.1.2 External connectors

For displaying audio and video on a TV, the set-top box is equipped with standard scart and s-video connectors. Also, RCA connectors are available for mono and digital 5.1 stereo sound.

RJ45 connector for 100Mbit network is available and used for both receiving of content and download of the set-top box software.

There is also a possibility to connect external hardware through a USB1 interface available on the box.

3.2 System hardware

The set-top box today is a product of design that prioritises small size, low energy consumption and relatively low price. This has several effects on the box:

- The box is quite small, i.e. no place for a hard drive in the current design.
- The shell is not designed for hot and noisy elements.
- The box is quite cheap compared to external hard drives.
If we take a look at the possible ways of extending the system to make recording possible, the most obvious solution, adding an internal hard drive, is not appropriate because of the reasons above. Possible hardware solutions are:

1. Using external hard drive connected to the USB bus or the ethernet port.
2. Using a stand-alone NAS\(^1\).
3. Using a PC in the local area network as NAS and writing appropriate software for this system component.

External hard drive is costly, but appealing because of its simplicity. Also, some users may already have an external hard drive at home which makes this a viable solution.

The NAS solution is unfitting for two reasons. Most NAS units are quite expensive for end users and using such would mean increased complexity in the set-top box software. This is not a good solution at the moment, but might be in the future.

Most users have a PC at home so using it as storage is also quite appealing. This is definitely a solution worth looking into. Thus, both PC-based and external hard drive solution should be tested.

3.3 Software

Figure 3.2 shows the system architecture. The top layer shows user applications. Below this layer there are services and at the bottom we have the operating system and the set-top box hardware.

3.3.1 User applications

These applications allow the user to interact with the system. User commands are sent by remote control or wireless keyboard.

\(^1\)Network Area Storage. A stand-alone unit, connected to the local network which provides storage services.
For example, the user watches the TV and changes channels through the TV application, changes settings with a settings application and chooses active application using the navigator application.

### 3.3.2 Services

In the middle layer we find the services. These are used by the user applications or other services to perform common tasks. For example, the media service and the video mixer are used for playback of audio/video streams. The info service is used for saving and retrieving permanently stored data, such as the user settings.

### 3.3.3 The operating system and the hardware

The operating system provides system calls and libraries for convenient programming. These are used both by the services and the user applications.

The hardware is abstracted by the hardware abstraction layer (HAL). This includes common hardware and the dedicated video-decoding hardware.

### 3.3.4 Important modules

System software which is of particular interest for this project is presented in figure 3.3. These modules are used for playback of streams and for user control of what is shown on the screen.

As can be seen in the figure, the important code modules are:

- **Streamer** – video/audio stream player
- **Media manager** – software for managing a set of streamers
- **User application** – software for user interaction

If we take a look at the architectural view of the system, we find the Media manager in the service layer and user applications in the top layer. The streamer can be considered to be a service layer component.
Current system state and requirements

19

Figure 3.2: The system software divided in layers. Modules of particular importance for this project are coloured pink.

Figure 3.3: The parts of the system responsible for playing streams. Dashed arcs are process borders and arrows indicate usage or data-flow.
Recording was not taken into account when the software was written and parts of it should be redesigned for best solution to given problems. This is discussed in next chapter. Below, the software modules mentioned are described in more detail.

### 3.3.5 Streamer

This system component is responsible for processing media streams, e.g. separating audio and video and presenting those to the user via dedicated hardware.

Stream processing is done in the elements, starting with the source-element. These are represented by squares in *streamer1* in figure 3.3. Each element in the chain processes the stream in some way such as: dropping the data, modifying the data, splitting the stream into sub-streams or sending the stream to the hardware.

*Streamer core* is the part of the streamer which is responsible for providing services for the elements and controlling the elements when this is requested by the *Media manager* or some other software component. It also notifies elements of external changes when this is relevant for the element functionality.

Since the streamer is where all stream manipulation is done, it is the natural place for a main part of the recording functionality. This functionality will be added by inserting a new element into the streamer.

### 3.3.6 Media manager

Several streamers can coexist in the system and this is the module that keeps track of them. *Media manager* monitors streamer states and initiates state transitions when this is necessary. Example of streamer states are: *playing*, *stopped*, *rewinding* etc. *Media manager* also controls which streamer is active and thus has access to the set-top box hardware.

### 3.3.7 User application

These are the applications the user interacts with, for instance the TV application which controls what the Media manager presents, according to
users requests.

For a general description of user applications, see section 3.3.1.

### 3.3.8 URI loader

Another important software module is the **URI loader**. It is a system service and it is shown in figure 3.2. This service can be used by any user application when the application needs to handle a URI of unknown type. Then the **URI loader** can be used to activate another application, capable of handling this type of URIs.

### 3.4 Requirements

Previous chapter gives insight into how large this area really is. Making a really good PVR takes a lot of time, and this project is limited to 20 weeks full-time work, including writing the report. Therefore the focus will be on implementing the basic recording functionality and thus test how well the system can be extended to support recording.

Requirements are divided into two categories, **basic** and **extra** requirements. The **basic** requirements must be met and one or more of the **extra** requirements will be met if there is time at the end of the project.

Requirements are listed in table 3.1 on page 22.
## 3.4. Requirements

<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start/stop instant recording by pressing a button on the remote control.</td>
<td>basic</td>
</tr>
<tr>
<td>2</td>
<td>Store the recording on an NFS mounted file system.</td>
<td>basic</td>
</tr>
<tr>
<td>3</td>
<td>Store the recording on a PC in a Local Area Network.</td>
<td>basic</td>
</tr>
<tr>
<td>4</td>
<td>Store the recording on an external hard drive (USB 1.1).</td>
<td>extra</td>
</tr>
<tr>
<td>5</td>
<td>Pause live stream and continue playing later.</td>
<td>extra</td>
</tr>
<tr>
<td>6</td>
<td>Rewind and fast forward in a time-shifted stream.</td>
<td>extra</td>
</tr>
<tr>
<td>7</td>
<td>Record a stream that is not shown on the tv (background recording).</td>
<td>extra</td>
</tr>
<tr>
<td>8</td>
<td>Write a user-level application for scheduling recordings and managing recordings in progress.</td>
<td>extra</td>
</tr>
<tr>
<td>9</td>
<td>Protect stored content which is not free for further distribution. Playback should only be possible on the set-top box that recorded the content.</td>
<td>basic</td>
</tr>
<tr>
<td>10</td>
<td>Look into possibilities for using ECG to schedule recordings.</td>
<td>extra</td>
</tr>
<tr>
<td>11</td>
<td>Schedule recordings with parameters channel, start time and end time.</td>
<td>extra</td>
</tr>
</tbody>
</table>

Table 3.1: Requirements for the extended system.
Chapter 4

System architecture

First, some architectural models are discussed. Later in this chapter, architectural models are chosen for the streamer and for the overall system solution. Implications to these choices are discussed afterwards.

4.1 Extending the streamer

Since most of the work on this thesis will be extending the streamer, streamer architecture is also the most important architectural decision to make. In following sections streamer models are discussed and one of the modules is chosen for this project. These models show only the part of the element chain where the recording element is inserted.

4.1.1 Extended streamer models

There are several possible models which the extended streamer can match. These models affect the set of implementable features in different ways and relate to the requirements accordingly.

In figure 3.3 the original streamer is depicted. A transport stream flows through the streamer elements after it has been read into the first element in the chain. Finally the stream is consumed in some way, which for most
of the stream equals playback in the set-top box hardware.

Figure 4.1: Both pictures show streamer models capable of simultaneously writing to a permanent media and playing the stream on the set-top box hardware. To the left is a streamer capable of using file i/o to write to locally mounted file-systems and to the right is a streamer which uses Unix sockets to write a stream over a LAN.

4.1.2 Extending with elements capable of writing

A peek at the requirements shows that basic requirements 2, 3, 4 and 9 must directly influence streamer architecture. To the left in figure 4.1, is a streamer model that supports writing to a file whereas to the right is a model which supports writing to the network. These models are compatible with requirements 2 (store NFS) and 4 (store USB), resp. requirement 3 (store LAN).

The figure shows streamers when the stream is both displayed to the user and saved on storage. It is of course possible only to save the stream on the storage or only to display the stream without saving it.

To the left in figure 4.2 is a model which represents a streamer capable of storing the stream on both local hard drive and on LAN. It will be used when storage type isn’t the issue.

To the right in this figure is a similar model, with the difference that it supports encryption and decryption of streams. This model is also compatible with requirement 9 (content protection).
4.1.3 Properties of write-elements

As we can see above, the model to the right in figure 4.2 is compatible with all basic requirements, and is hence a viable solution to the problem.

This model is however far from perfect, which is noticed if requirement no 5 (time-shift) is considered. The only way to time-shift a stream is to use one streamer without output to record it and another streamer to continue playing the stream at user request.

This would increase complexity in the system since now not only one, but two streamers must be associated with current playback. Many playback problems would also be noticeable, such as:

- Longer times between user action and when stream really starts recording.
- Stopping time-shift would lose a couple of seconds while shifting to live stream. This happens because of the transition between the streamer playing the live stream and the streamer which is playing from storage.
- Forwarding in a stream would cause problems when end of time-shift file is reached. This is similar to the problem above.
These problems are very similar to the problems I encountered while testing the Force box, see section 2.3.3, which probably shows that their solution is similar to the one I just described.

### 4.1.4 Extending with elements capable of writing and reading

To circumvent the problems described in previous section, the streamer can be extended with an element which is capable of both reading and writing from permanent media. This is depicted in figure 4.3.

Figure 4.3: A model of a streamer capable of reading from and writing to a permanent storage. The figure shows the case when the stream is time-shifted, but this streamer can also write to the storage while displaying the real time stream to the user.

This model fits all requirements which are compatible with write-elements, but also gives the possibility to time-shift a stream using only one streamer process. In such a streamer seamless transitions between streams should be possible, which is necessary for good time-shift functionality.

Problems with this solution is that current design only supports continuous stream flow, i.e. it is not possible to break the flow by inserting an element that both reads from and writes to a permanent storage. Making these changes means making extensive modifications to the streamer core. Using this functionality from other parts of the system implies also extending these parts, i.e. additional effort to make everything work.
4.1.5 Conclusion

Extending the streamer to support reading and writing to permanent storage would demand much design work with both the streamer element and the streamer core to support discontinuities in the stream flow. This would probably take a large part of the 20 weeks to which this project is limited. This is in no way compatible with the goal of testing the system and this is obvious when taking a look at the basic requirements. Therefore, the streamer will be extended with a write-element such as depicted by figure 4.2.

4.1.6 Content protection

Much of the content broadcasted today is of a type that needs to be protected from free copying and further distribution. This can be content from non-free channels or, in digital television, video on demand.

There are several “standards” for content protection, used by different content providers. A good solution for content protection on storage would have to be an extension of an existing protection system, i.e. special solution depending on the content provider.

It is far out of scope of this thesis to implement a good content protection system. Rather, this implementation is supposed to provide a realistic test bench for encryption on this set-top box system.

Encryption model

A model for implementing a basic content protection system is depicted in figure 4.2. A stream is encrypted before it is written to storage and all streams pass through a decryption module during playback. This means that encrypted streams must be marked as such, to make decryption possible.

Algorithm

AES, Advanced Encryption Standard is a symmetric\(^1\), common purpose encryption algorithm. It is used in other parts of the set-top box which

---

\(^1\)Encryption and decryption of data is done with the same key.
4.2 Extending the system architecture

To make use of the new, extended streamer it is necessary to extend the overall system architecture. Figures 3.2 and 3.3 show the current state of the system.

The general idea is to add a user-level application to handle recording. This application will be accessible to the user directly or via other user-level applications such as the TV application. Communication between the pvrdk which is our new application, and other user-level applications will go through the URI loader.

This is shown in figure 4.4. Here, services are represented by squares, user applications have slightly rounded corners and streamer boxes are almost oval. This figure should be compared to the architectural view in figure 3.2.

This new system view shows only important software components and usage is shown with arrows. Dashed arrows show how the URI loader is used to invoke other applications in the system.

4.2.1 PVR demo

PVR demo is a user-application which will be running in the background in order to shorten response times when either:

- The user wants to switch to the PVR demo or

\(^2\text{Changing the key requires editing the source code.}\)
Figure 4.4: Updated architectural view of the system. User-level application PVR demo is added and connections between the user-level applications and the streamers are shown in more detail. Dashed arrows show interaction between user-level applications.
Another user-application invokes the PVR demo through the URI loader.

In first case the user might want to see what’s being recorded at the moment or wishes to schedule a recording for later execution, in accordance with requirement 11 (schedule recordings).

The other way of invoking PVR demo can probably be put to much use, but one of the most important ones is to be able to receive commands for instant recording (requirement 1) from the TV application.

For playback of recorded files the VoD application will be used. This application is normally used for playback of video-on-demand, but should work just as well for playback of recorded programs.

Again, means of communication between PVR demo and the VoD application will be the URI loader, also this depicted in figure 4.4.

4.2.2 TV application

To make instant recording work, some small changes in the TV application are needed. This includes recognising the recording button on the remote control and invoking the PVR demo via the URI loader.

4.2.3 VoD application

This user-application is used to play video-on-demand content, but can also be used to play content from a local hard drive. PVR demo will use the VoD application to play stored files. Invocation is done via the URI loader, also this shown in figure 4.4 on page 29.

4.2.4 Interfaces

Except for the most obvious changes, some other modifications to the software are necessary, such as extending the interfaces to some modules to support recording. Figure 3.3 on page 19 shows quite well that changes are needed in interfaces between user-applications and the Media manager and between the Media manager and streamers. Functions to start and stop recording will be added to these interfaces.
4.3 Implications to the chosen system solution

There are actually two solutions since the streamer, and hence the system, will support writing to both an USB disk drive and to a computer on the local network. Accessing the files stored on a local file-system can be done through standard libraries, but access to the files on LAN is a bit trickier.

4.3.1 USB disk

As mentioned above, access to local files is not a problem since the streamer has support for using local files as a source. Still, browsing local directories and such must be supported by the PVR Demo.

4.3.2 Network storage

The best solution to network access would be extending the system to be able to browse and read files via same protocol that is used for storing remote files. However, if this isn’t possible, other means of accessing these files exist:

- Broadcasting the files from the remote host.
- Making files accessible via protocols already supported by the box, such as http or rtsp.

This issue is discussed further in the following chapter.
4.3. Implications to the chosen system solution
Chapter 5

Design and Implementation

This chapter describes the overall design of the system which will be used for recording. Design of some specific software modules is described in detail. First however, some design issues and choices are presented.

5.1 Modules

A home system capable of recording could look as shown in figure 5.1. Recorded streams can be saved on the USB hard drive that is directly connected or on a PC in the local network.

Depending on which solution is used, different software is needed for a functioning system. A streamer element is necessary in both a hard drive and a PC based solution. This new streamer element will from here on be called *diskwriter element*. Note that a diskwriter element for the PC based solution is much more complicated. Saving streams on LAN also requires server side software if such doesn’t exist for the chosen protocol.

Also, the interfaces in the system as well as the TV application have to be slightly modified. Application PVR demo has to be written. Thus,
Figure 5.1: Conceptual view of a recording capable home system. The set-top box can record streams both to a directly connected USB hard drive and to a PC in the local network.

The following software modules have to be in place for recording to work properly:

- Streamer interfaces
- TV application
- PVR demo
- Diskwriter element

And in case a remote PC is used to store recordings:

- Server side software
- Diskwriter element (capable of writing to LAN)

Before going into details of the design, decision regarding the protocol between the set-top box and the PC on the LAN, must be made. This is done in the following section.
5.2 Design issues and decisions

5.2.1 Protocol between the set-top box and the PC

The box is an embedded system, and the diskwriter element should be kept simple for several reasons. The most important ones are: the set-top box software is hard to debug compared to software on a PC and the performance is very important.

Since the system is booted of the network, the entire root disk\(^1\) is downloaded at start up. This causes problems since installing new libraries makes the download slower and takes space in the RAM after start up.

This sets some restrictions on the choice of the protocol used between the set-top box and the PC.

Further, following should apply to the protocol in question:

1. Good portability for both client and server implementation of the protocol.

2. It should be as easy as possible to test and debug the protocol, the client and the server.

3. It should be easy to add features to/extend the protocol.

4. Possibility to preprocess the data before it is written to the disk on the PC. This can be used for modifying the stream, e.g. to correct the stream if errors occur during transfer.

5. If open source software is used, the software must be licensed under LGPL or a similar license that does not require revealing source code for the set-top box.

6. Implementation of server and client software should take as little time as possible, i.e. the protocol should be simple.

\(^1\)Compressed set-top box file-system.
5.2.2 Existing protocols

There are at least two existing protocols that might be good enough for communication between the set-top box and a PC server, http and rtsp. These protocols and their properties are discussed shortly below.

HTTP

HTTP, or the hypertext transport protocol is the common protocol of the WWW and supports both reading and writing of files through methods GET and PUT. Most, if not all http servers also support directory listing\(^2\) which is enough for implementing at least a primitive remote file system.

Advantages of using an existing protocol such as http are:

- Suitable server side software already exists and there might exist libraries which can help in development of the client software.
- Using a standard protocol makes the set-top box easier to use and integrate into existing home networks.

This is a project limited to 20 weeks and this is a big factor which must be taken into consideration when choosing design. In light of this the disadvantages of choosing http for the communication can be viewed.

- Not having full control of the software means it is harder to make changes in latter stages of the project. Big changes might make the chosen protocol unfit making some or a lot of the software unusable.
- Implementing the client on the set-top box might be harder than necessary which means added complexity to the set-top box code.
- It is tricky to estimate the time needed to implement the client, which means several weeks might be spent only on gaining this insight. Since this project has the above mentioned time limit, this could be a serious setback.

\(^2\)Browsing the content on the server.
RTSP

RTSP is abbreviation for real-time streaming protocol and its name describes well what it is used for. This protocol could be used for transfer of files between the set-top box and the file server, just like the hypertext transport protocol.

Besides the advantages and disadvantages of the http, rtsp is a protocol which was created for streaming media which makes it very suitable for playback of streams from the server. Listing directories and saving streams on the server might however be harder then in the http case, but this could be solved by combining rtsp with a complementing protocol.

5.2.3 Protocol choice

From a user perspective an existing, widely used protocol, would be a perfect choice. However, if we take a look at the restrictions list on page 35, we can see that using an existing protocol doesn’t go well with the protocol requirements: 3 (extendibility), 4 (preprocessing of data) and 6 (simplicity). There is also the issue of predictability which is discussed in section 5.2.2. Designing and using a new protocol is at the moment both good enough for the project and the most predictable alternative considering development time. This choice is a natural one, also because of the main goal of this thesis work which is to test the set-top box choosing simple, yet not restrictive solutions.

Therefore, a simple, but sufficient protocol for file transfer will be designed and implemented for use on the set-top box and the server PC.

Except for the protocol requirements, some choices I have made will influence the design decisions and some implementation details. These are:

- Common parts of the protocol and the server implementation will be written in C [5].

- The server will be single threaded, using the Unix system call select().\(^3\) Select makes it possible to handle multiple connections to the server without making the server multi-threaded. This choice is

---

\(^3\)Used for detecting changes on file descriptors without polling. See manual page for select(2) on any GNU Linux system.
made for educational purposes, since these concepts probably will be very useful in further system development.

RFSP, the *Remote File System Protocol*, is the name of the protocol that will be designed for use between the set-top box and the PC on the LAN. The decisions below map to the protocol requirements on page 35 and the RFSP will be designed in accordance with these.

1. Implement in C. The code should be reasonably portable and the common protocol code can easily be used on the client side since the streamer is written in C++ [6].

2. Clear text protocol to make debugging and testing reasonably easy. This makes it much easier to test for example the server, before the client is written.

3. Since the protocol is written from scratch, it will of course be designed for extensibility.

4. Code will be designed in a manner that makes it easy to process data before writing to disk or after reading the data from disk.

5. There will be no licensing problems since only the standard libraries will be used.

5.3 Remote file system protocol

This section will only describe the protocol and not any implementation of it. Some implementation-specific details might end up in this chapter, but that is only as a side effect to describing RFSP. RFSP is designed for this thesis work for transfer of files between the set-top box and a server on the local network.

RFSP is a clear text protocol, such as telnet or FTP. Similar to the FTP, one connection is used for communication between the client and the server and separate channel(s) are used for transfer of data. For the communication channel, the TCP is used but the data can be transferred via both the TCP and the UDP. In the first implementation only the TCP will be supported, also for file transfers.
5.3.1 Commands and responses

Commands and responses are sent via the command channel. The syntax for commands is:

\[ <\text{cmd}> [\text{<param1>...<paramn>}] \]

However, commands can have one or several required parameters. Each command is described in more detail in following subsections. For a complete list of commands and brief descriptions, see table 5.1 below.

Responses have following syntax:

\[ <\text{response}> <\text{response code}> <\text{response desc}> [\text{response data}] \]

where \(<\text{response}>\) is OK or ERROR. Response might from here on be referred to as error and response parameters might be referred to in similar way (e.g. error code in stead of response code).

Response description and response data are strings starting and ending with a double quotation mark. A backslash is the escape character and works in a common manner.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Response description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELLO</td>
<td>Used for checking whether connection was made to an RFS server.</td>
<td>Id string.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Open a file by name.</td>
<td>File id or error.</td>
</tr>
<tr>
<td>close</td>
<td>Closes a file by file id.</td>
<td>Success or error.</td>
</tr>
<tr>
<td>PWD</td>
<td>Get working directory.</td>
<td>Current working directory.</td>
</tr>
<tr>
<td>CD</td>
<td>Change directory.</td>
<td>Success or error.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Change file mode to writing.</td>
<td>Success or error.</td>
</tr>
<tr>
<td>QUIT</td>
<td>Quit the command session.</td>
<td>Server closes the TCP session.</td>
</tr>
<tr>
<td>READ</td>
<td>Change file mode to reading.</td>
<td>Success or error.</td>
</tr>
<tr>
<td>ERROR</td>
<td>A response to the last command.</td>
<td>Indicates error.</td>
</tr>
<tr>
<td>OK</td>
<td>A response to the last command.</td>
<td>Indicates success.</td>
</tr>
</tbody>
</table>

Table 5.1: Commands and responses for the RFS protocol, version 0.1.
Both commands and responses end with a \n. A command or a response not terminated with a \n will cause a parse error when next command or response is received.

**HELLO**

Says hello to the server. This can be used to check if the connection is ok.

Syntax:

```
HELLO
```

Responses:

```
OK 002 "Request for info succeeded" "RFS Server, <version>"
```

**VERSION**

Ask for the version of the server.

Syntax:

```
VERSION
```

Responses:

```
OK 002 "Request for info succeeded" "<version>"
```

**OPEN**

Open an existing or a new file on the server. Default file mode is write. On success an URI is returned where server port is the file-id.

Syntax:

```
OPEN <filename>
```

Responses:

```
OK 003 "File opened successfully" "protocol://server-ip:port"
ERROR 011 "Path exceeds maximum path length"
ERROR 012 "Tried to open something that is not a file"
ERROR 013 "Permission denied"
ERROR 014 "Can’t open file for unknown reason"
ERROR 016 "Server is to busy, try again later"
```
CLOSE
Closes an open file.
Syntax:
    CLOSE <file-id>
Responses:
    OK 004 "File closed successfully"
    ERROR 015 "No file associated with this file-id"

PWD
Get the working directory from the server.
Syntax:
    PWD
Responses:
    OK 002 "Request for info succeeded" <current working dir.>

CD
Change working directory.
Syntax:
    CD <path>
Responses:
    OK 005 "Directory changed successfully"
    ERROR 010 "No such file or directory"
    ERROR 011 "Path exceeds maximum path length"

WRITE
Change file mode to writing.
Syntax:
    WRITE <file-id>
Responses:
    OK 006 "File mode changed successfully"
    ERROR 015 "No file associated with this file-id"
Note: not implemented. When file is opened, file mode is automatically set to write.
4.3. Remote file system protocol

QUIT

Asks the other part to close the command session.
Syntax:
    QUIT
Responses:
    The TCP session is closed.

READ

Change file mode to reading.
Syntax:
    READ <file-id>
Responses:
    OK 006 "File mode changed successfully"
    ERROR 015 "No file associated with this file-id"

Common responses

There are some responses that can be returned as an answer for any command. These are described below.

    ERROR 007 "Syntax error" – incorrect syntax for this command
    ERROR 008 "Illegal parameter" – one/several invalid parameters
    ERROR 009 "Unknown command" – unknown/unimplemented command
    ERROR 017 "Internal server error, please consult the server logs" – internal server error occurred due to error in the server.

5.3.2 Connecting to an RFS server

As mentioned earlier, separate channels are used for commands and data. One TCP connection is used for client/server communication plus one additional connection per open file on the RFS server. At the moment, only normal TCP connections are supported for file transfers.
**Initial connection**

To start communication, the client connects to the server on some default port which is not decided at this moment. Server may or may not identify itself upon connection. If the server identifies itself, the response must be the same as the response given to the `HELLO` command.

If the server doesn’t identify itself, the client should ensure that it is an RFS server to which it connected and that the client is compatible with the server version of the protocol. This can be done using commands `HELLO` and `VERSION`. Figure 5.2 shows this client/server interaction.

![Diagram of client/server interaction](image)

**Figure 5.2**: Shows interaction between an RFS client and an RFS server. First, the client connects to the server and then it opens a file for writing.

**Reading and writing files**

When connection is established, files can be opened with `OPEN` which gives the client a new URI pointing to the opened file. This URI looks like:

```
protocol://host[:port][path]
```

Our implementation of the protocol only supports URIs of type:

```
tcp://host:port.
```
This URI can be used to open a TCP session to the file, in much the same way as connection is made to the RFS server in first place. Default file mode will be write but this can be changed to read via the communication channel. Details for OPEN, and other necessary commands can be found in section 5.3.1.

5.3.3 A short example

The example below shows what an interaction with an RFS server can look like. Here, nc⁴ is used to connect to an RFS server whereupon some commands are sent to the server.

```
shell% nc nisse 5555
HELLO
OK 002 "Request for info succeeded" "RFS Server, v0.1"
VERSION
OK 002 "Request for info succeeded" "0.1"
OPEN test.file
OK 003 "File opened successfully" "tcp://192.168.51.10:5557"
CLOSE 5557
OK 004 "File closed successfully"
CLOSE 1212
ERROR 015 "No file associated with this file-id"
PWD
OK 002 "Request for info succeeded" "/
HELLO world
ERROR 007 "Syntax error"
FOO
ERROR 009 "Unknown command"
```

Observe in particular the command OPEN. After successful opening of file “test.file”, another nc session could have been used to connect to the file and write to it. If we wish to read the file, READ command must be issued before connecting to the file URI.

---

⁴"net cat" – used to establish TCP connections from the command line (on Unix systems).
5.4 RFS server

As decided earlier in section 5.2, the server has to be extendible (easy to add new commands/responses) and reconfigurable (e.g. easy to add a preprocessing function for data that is written to disk). Also, different sessions have to do different things with the incoming data, e.g. file transfer session and command session have to save data to disk respectively parse client input.

This is solved by dividing the server into suitable modules, as shown in figure 5.3. The modules are:

1. server loop
2. data parsers
3. command parsers
4. helpers

Figure 5.3: RFS server can be viewed as the depicted modules, when data parsing is “used as a divider”. Note that only the command session uses command parsers, but all types of sessions use a data parser.

5.4.1 Server loop

The server loop is responsible for keeping track of the sessions. This includes following:
• Keeping track of which sessions are active, i.e. if data is available for reading or writing.

• Reading data from TCP-sockets when data is available and calling correct data parsers.

• Writing data to socket when the write buffer for a session isn’t empty.

Described shortly, server loop assures data flow to and from the server. It also keeps track of sessions and initiates data parsing when the read buffer(s) are updated. See also figure 5.4.

Figure 5.4: depicts how data in a command session is parsed and what kind of data each module receives/reads.

5.4.2 Data parsers

When data is read into a buffer, a previously registered data parser for this session is called. The parsing function does different things depending on the session type, as can be seen in the table below:

<table>
<thead>
<tr>
<th>Session type</th>
<th>Data parser description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server session</td>
<td>Waits for incoming connections and starts new command sessions. Removes leading white spaces, looks for \n and command name and calls the correct command parser. Writes all incoming data to a file associated with this session, or reads all data from a local file until buffers are saturated.</td>
</tr>
<tr>
<td>command session</td>
<td></td>
</tr>
<tr>
<td>fileio</td>
<td></td>
</tr>
</tbody>
</table>
Since a data parser is associated with each session, sessions can change type and new ways of parsing data can be incorporated into the design easily. As shown, the data parsers don’t have to strictly parse data, but can do what is needed by the session they belong to.

Data parsers for the server session and fileio are pretty straightforward, but the parser for command session is a bit more complex.

When new data is in the read buffer, it is parsed and the command is looked up in a command table\(^5\) After this, the correct command parser is called for each complete command. See also figure 5.4.

### 5.4.3 Command parsers

Command parsers implement most of the practical details server is asked to do. This includes: opening and closing of files, keeping track of and changing working directory, initiating closing of sessions when requested, checking validity of the parameters and creating responses which are sent back to the client, etc. For more details, see the source code [7].

### 5.4.4 Data structures

Session has been mentioned several times in this chapter, and there is more to it then it might seem at first. Also command table has been mentioned, but there are several other data structures which have not been discussed yet.

The most important data structures in the server are:

- session
- session-set
- command table
- response table

\(^5\) This table holds command names and parsers for each command.
5.4. RFS server

Session

This is a complex and central data structure used by both the server and the client. Each session holds a socket and how this socket is used decides the type of the session. Example of such sessions are server session, client connection session and open file session. Each session structure holds much data, besides the session type. Description of other “session structure members” follows:

- TCP socket
  
  Connection to a local TCP stack.

- Session type
  
  Indicates which kind of session this is.

- Buffers
  
  Read/Write buffers for interaction with the client, indices into buffers, etc.

- Data parser
  
  Function pointer to a data parser for this session.

- Command-specific data
  
  Data that must be remembered between sequential uses of this session. E.g. data used by OPEN, CLOSE, CD etc.

Sessions are used all the time and are passed as argument to both data and command parsers.

Session-set

This is a data structure that holds all sessions and there is one instance of session-set per server. It also holds data needed by select() and information about the server such as hostname, ip and server port.

Command table

This table is used when an LF\(^6\)-terminated string is read by a data parser (in a command session) to find a match for the command just received by

\(^6\)Line Feed, usually produced by the ENTER key on Unix systems.
the server. If no match is found, a parser for a special command unknown is called.

**Response table**

The *response table* holds response IDs, names and descriptions. It is used by the server to create responses to client commands and can be used by the client to get a description for a response instead of parsing the string received.

**5.4.5 An example**

In figure 5.5, a not-to-detailed example of how the server works is depicted. Only major function calls are shown and nothing of the server start up is. For more detailed information about the RFS server, see the source code [7].

**5.5 Diskwriter element**

This code module is one of many loadable code modules that will be loaded in the streamer for stream processing. As such, there are some restrictions on this element which will influence the solution to this problem. The most important ones are:

- The streamer is single threaded (and therefore)
- All I/O must be nonblocking [8]

How this influences the implementation of the diskwriter element is described for each element code module. The structure of the diskwriter element is shown in figure 5.6.

Notice that this diagram represents an element capable of writing both to an USB disk drive and to an RFS server. Writing to a local disk is much simpler and code for doing this is placed in general class TDiskWriterElement. All other classes deal with writing to an RFS server.
Figure 5.5: Shows an example of server program flow without too many details. Time flows from top to bottom. Arrows represent function calls and module/function names can be found in bold style at the top.
Figure 5.6: UML diagram for the diskwriter element. Most important classes, outside of the element but important for it’s description, are also shown.
5.5.1 TDiskWriterElement

This is the main class in the streamer element. It is via this class that communication with the streamer core is done and the method \texttt{Process()} is called when there is data for this element to process.

This element is not doing any actual processing of the stream so the streamer core is notified immediately that the data is ready for processing by the next element. Between this instant and the next call to \texttt{Process()}, the stream must be saved to disk or network, lest it is thrown away.

**USB disk**

Saving streams to a USB disk is simpler than saving to an RFS server. When \texttt{Process()} is called, disk access functions from the standard library are used to write data. The code assumes that disk is available, thus making code even simpler.

\texttt{DiskWriterElement} also keeps track of recording state through a simple state machine. This state machine is a reduced version of the state machine in figure 5.7. It consists of states \texttt{w\_OFF} and \texttt{w\_ON}, with transitions on stimuli from the \texttt{Media manager}. On error, a transition to \texttt{w\_OFF} is made.

For further discussion about writing to a local disk, see chapter 7, page 65.

**RFS**

If the data is saved to the network, \texttt{TRfsClient} is used for connecting to the RFS server and for transferring the stream over the network. Since this involves operations which can take an unbounded amount of time, the code can not be blocking. Therefore, keeping track of the program state has to be done with some kind of a state machine.

The state machine used is shown in figure 5.7. There are four states: \texttt{writing\_off}, \texttt{writing\_connecting}, \texttt{writing\_openingfile} and \texttt{writing\_on}. These states are tightly connected to the RFS protocol and to what the client needs to do before data can be written to a file on the server. The state machine in figure 5.7 shows also the events which cause state transitions.
Figure 5.7: A state machine describing recording state of the TDiskWriterElement. W_OFF and W_ON means writing is on resp. off. The other two states represent where in the connection process the client is, before everything is prepared for reading/writing.

5.5.2 TRfsClient

Communication with the server is done with an instance of this class. All I/O in this class has to be nonblocking which means the element must be notified of activity on file descriptors it is using. System classes IEvent and IEventCallback are used for that purpose, see section 5.5.3 below.

Since all I/O has to be nonblocking, the return values of the functions can not be used in regular manner and indications of progress or failure must be implemented with callback functionality. This is described more in detail in section 5.5.5.

What happens is that after a call to a nonblocking function, success is returned to the caller (hopefully) and action is taken. This can for example be an attempt to connect to the server. After a certain time period, the call will succeed or time out. The result is written to response variables and the caller is notified of this via the callback interface. Then, it is up to the caller to read the response to its latest action.
5.5.3 IEvent and IEventCallback

The IEvent provides functions for registering file descriptors for read, write or a combination of these events. This interface is provided by the streamer core.

The IEventCallback interface is implemented by a class which uses the IEvent and wants to get notifications of events on registered file descriptors.

These interfaces are a simple front-end to a system select()-loop. It is up to the user of the interfaces to keep track of registered descriptors and event masks\(^7\) so trying to register or unregister events on a descriptor, when this already has been done, results in a program abort. This problem is the reason for creating the IEventManager, which is described below.

5.5.4 IEventManager

This class adds functionality to the IEvent-interfaces. The extra functionality consists of:

- Completely transparent registering and unregistering of file descriptors. Previous state of the file descriptor is indicated only by the return value.
- Extra data can be stored with registered file descriptors.
- Possibility to query the mask of a file descriptor.
- In case of program failure or similar, all file descriptors can be unregistered easily.

5.5.5 TResponse and IResponseCallback

These code modules work in almost the same way as the IEvent and IEventCallback. Implementing IResponseCallback means that functions OnConnect and OnResponse must be defined and this is done by TDiskWriterElement.

\(^7\)A binary mask is used to decide which events select should listen to.
Inheriting from the `TResponse` gives access to functions which can enable and disable callback. This is done by the `TRfsClient` which also gives the creator of the client possibility to enable or disable the callback.

Reason for using these classes is to avoid status polling from the `TRfsClient`. When response classes are used, RFS client can notify another class that a non blocking connection has returned or that a response from the RFS server is available for reading.

The UML diagram in figure 5.6 shows how these classes/interfaces fit into the diskwriter element.

### 5.5.6 Content protection

**Transport stream**

Streams played by the set-top box are most often in *transport stream* (TS) format, which is audio and video multiplexed into a single stream. The TS is divided into *transport packets*, each 188 bytes long, with specified header and optional data payload [9].

Before the stream reaches the `DiskWriterElement`, it is divided into *transport packets*. Transport packets plus some additional data is bundled into *segments*. One such segment is depicted in figure 5.8.

![Figure 5.8: A segment with three transport stream packets. Dashed lines separate frames.](image)

**Encryption of packets**

Transport packet header holds information about the rest of the packet. Amongst flags in the header, `transport_scrambling_control` [9] can be found.
This is a two-bit flag indicating whether the packet payload is scrambled or not.

When AES is used to encrypt content, only the packet payload, i.e. audio/video data will be encrypted. This means that the previously mentioned flag, transport_scrambling_control, can be used to mark and identify encrypted packets.

Decryption of packets

When content protection is enabled, encrypted packets will be identified and decrypted before they are sent to the next element. Unencrypted packets will simply be forwarded to the next element.

5.6 PVR demo

Writing this application will mostly be about writing a GUI\(^8\). All C++ applications on the set-top box must be written with the GTK 1.2 widget set [10]. For more details on GTK, see the GTK web page, http://www.gtk.org/.

Since the GUI is the only interesting aspect of this application, it will also be the only thing described in this document. For details about the PVR demo see the source code for this application [11].

5.6.1 Features

Through this application, the user should be able to do following:

- See what is being recorded at the moment.
- List old recordings.
- Start playback of old recordings.
- Schedule new recordings.

\(^8\)Graphical User Interface.
• Start instant recording of the active TV channel. (Indirectly, through the TV application.)

Of these features, seeing what’s being recorded and indirect start via the TV application will be implemented. Other features should be implemented and will be if there is enough time.

5.6.2 GUI

This application has to display many things, and everything won’t fit on one screen. Therefore, a menu will be the start point in the application. From here, the user will be able to navigate to other sub-screens which will be described below.

Sub-menu: status

This sub-menu will show what is being recorded at the moment and the hard drive status (free/occupied space, space occupied by current recording).

Sub-menu: view recordings

This sub-menu will show a list containing the stored recordings with possibility to play or delete these.

Sub-menu: schedule recordings

The “schedule recordings” sub-menu will allow the user to set the channel and the start/end time for a recording and to add this recording to the pending list of scheduled recordings.
Chapter 6

Discussion

This chapter summarises what has been done during this thesis work, results from benchmarks and problems with the chosen recording solution. It should be consulted before further work is done on this project.

6.1 From requirements to a solution

At the start of this thesis, the aim was set high. The goal was to implement a functioning PVR solution with recording and time-shift capabilities. A little voice of warning existed even then, but was easily ignored. Later, during the investigation phase, I realized the complexity of the system and the number of pitfalls that existed and that voice made it self heard again. This resulted in a more cautious set of requirements and care while making the architectural choices.

6.1.1 Requirements

Still, considering everything, much was done in only one semester. A look at the requirements shows that all basic requirements were fulfilled (requirements no: 1, 2, 3 and 9). Also, extra requirements 4, 7, 8 and 11 were implemented which only leaves extra requirements 5, 6 and 10 unim-
implemented. These unimplemented requirements deal with time-shifting of tv programs and scheduling with ECG. (See the requirements table 3.1 on page 22.)

6.1.2 The system

The extended system is now capable of recording streams, initiated both by a button press and a timer. A graphical user interface is in place and it allows the user to see what is being recorded, manage recordings and start playback of stored files.

This applies only to the USB solution, since the system isn’t capable of reading files stored on an RFS server. Playback of such files is done with an external application called *The Video LAN Client*, or VLC. VLC is capable of broadcasting MPEG2 streams [12].

6.2 Benchmarks

One of the side effects of implementing this solution is the knowledge that it is possible to do so. Before this thesis, it was uncertain how easy it would be to extend the software for recording and what impact this has on total system performance.

Benchmarks which are presented below are not terribly accurate, but are intended to give an overall view of the CPU usage with an extended streamer and bus usage when a USB hard drive is used as storage.

6.2.1 USB

The set-top box used in this project has only got a USB1 interface, which means the upper theoretical data transfer limit is 12Mbit/s. The streams in the test lab are usually around 5-6 Mbit/s, with transfer rate peaks as high as 10Mbit/s.

This could clearly be seen when trying to record a stream and simultaneously play another stream from the disk. The system usually works for a number of seconds and then behaves unexpectedly. A total freeze or
a system crash was the most common behaviour in these situations. The reason for this behaviour is described in section 6.3.2.

If a USB hard drive is used in any real products only one stream can be transferred to or from the disk at the time. This means that time-shifting or saving multiple streams to the disk would require a USB2 interface.

This problem doesn’t occur when streams are saved on 100Mbit-LAN, since the set-top box is equipped with a 100Mbit network card. In this case, the CPU is setting the limit to how many streams that can be saved/read simultaneously.

### 6.2.2 Encryption and CPU usage

Encryption in this test was done without analysing the stream; instead, every tenth or every hundredth packet was encrypted to get a rough feeling of how much encryption is needed to render a stream unusable for someone without correct decryption key. Of course, better results can probably be achieved if packets are strategically selected for encryption.

Encryption and decryption of data is a quite CPU demanding task. The set-top box that was used during this project is equipped with a CPU which in performance is comparable with a PII@200MHz. Table 6.1 shows how much CPU a modified streamer uses, depending on how much of the stream data that is encrypted. Values are approximate which really makes differences such as 2% in CPU usage quite insecure (as in the difference between CPU usage when 1% or 10% of the stream is encrypted).

CPU usage seems identical when the streams are decrypted, so table 6.1 can be consulted also when this information is considered.

<table>
<thead>
<tr>
<th>Encrypt(ed) data</th>
<th>Streamer CPU usage (approximate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>25%</td>
</tr>
<tr>
<td>1%</td>
<td>25%</td>
</tr>
<tr>
<td>10%</td>
<td>27%</td>
</tr>
<tr>
<td>100%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 6.1: Streamer CPU usage when stream encryption is used.
6.3. Problems with the current solution

<table>
<thead>
<tr>
<th>Enc. data</th>
<th>Viewing result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>Perfect.</td>
</tr>
</tbody>
</table>
| 1%        | Annoying distur~
| 10%       | The audio stream is completely destroyed. Lots of disturbances in the video stream. Objects are recognisable, but the total viewing experience is very low. |
| 100%      | No picture, no sound. |

Table 6.2: Viewing experience of encrypted streams.

The encrypted stream

It is interesting to see how much the stream is scrambled when it is encrypted and the correct decryption key is missing (table 6.2). Such tests were both performed on the set-top box and on a PC with MPlayer (http://www.mplayerhq.hu/). The results were very similar.

The conclusion is that encrypting 10% of the stream should provide enough protection for stored streams, especially if packets for encryption are chosen with more care than in this experiment.

6.3 Problems with the current solution

There are some problems with the implemented solution that must be taken care of should this project be developed further. The problems are described below and continuing discussion on some of the problems can be found in chapter 7, “Future work”.

6.3.1 USB interface

As mentioned above, a USB1 interface is slow to allow writing/reading of two streams simultaneously, which means time-shifting or saving two streams at the same time is impossible. This problem has to be solved if a USB hard drive is to be used as storage and the solution isn’t limited to only saving or reading one stream at the time.
6.3.2 File I/O

It seems that all file I/O is blocking, even if a non-blocking interface is used. This is a big problem, since the elements should not use blocking calls and failure to respond within a certain time interval means process termination.

Still, the current solution works quite well since the disk access usually is quite fast and predictable. It happens though that the system crashes, probably due to blocking file I/O calls. I suspect that this depends on buffering of data writing calls, so when the time finally comes to write accumulated data to the physical disk, the process is blocked for too long and is thus terminated by the system. This must be investigated in much more detail before continuing developing recording to a disk drive.

Deleting files

The above mentioned problem can be seen clearly when deleting large files\textsuperscript{1}. The set-top box runs Linux and uses ext2\textsuperscript{2} file-system, which at the moment is the only file-system available on the box. Deleting a recording of normal size can take a couple of seconds, which is above the maximum response time of the applications. This means that the application will be terminated when trying to delete a recording of normal size.

6.3.3 Recording awareness

When the recording solution was implemented, little concern was taken for the system to be aware of recording. A streamer element was implemented and the interfaces extended in order to be able to start recording at user request. Amongst problems with this approach are:

- No error reporting back to the caller in case of error.
- Media manager is unaware of the recording which means a special application is needed to handle recording (PVR demo.)

\textsuperscript{1}It takes noticeable time to delete files that are a couple of hundreds of megabytes large.

\textsuperscript{2}Ext2 is a commonly used, non-journaling file-system often used on Linux hosts.
Before continuing further development, the streamer and the Media manager must be extended in order to enable error reporting to the user or to the application which tries to use the recording functionality.
Chapter 7

Future work

7.1 Blocked disk access

In section 6.3.2 on page 63, the problem considering blocking file I/O is described. There are at least two ways of solving this problem, by modifying the Linux kernel to support nonblocking file I/O or by adapting the system by introducing a *File writing service*.

7.1.1 Modifying the kernel

This might be the best solution, but it is as well the most unpredictable one, unless one is very familiar with appropriate parts of the Linux kernel. This option needs further investigation.

7.1.2 File writing service

Another way of solving the problem of blocked writing is to create a service which must be used by applications that want to access the hard drive. This is more “going around the problem” than solving it, but it might be a feasible solution.

This solution would be multi threaded where one thread responds to system requests and the other thread is responsible for writing data to...
files. Data would flow between these threads through a FIFO, which would have a limited size. Requests for writing or reading, while this buffer is full, would not block but return error (*try again*), similar to other non-blocking interfaces in the Linux OS.

The blocking code would have to go through rigorous testing to avoid long down-times of this service. Still, the non-blocking thread would have to perform some kind of watchdog functionality so that the blocking thread is terminated and restarted in case of failure.

Also this solution demands much work since the set-top box stability would be compromised by solving the “blocking problem” this way.

### 7.2 Alternative streamer solution

In chapter 4, different models for extending the streamer are discussed. The model described in section 4.1.2 (extending with a write-element) is chosen for this project, whereas the other model is sacrificed on the altar of god of deadlines.

This section will briefly describe how the design of the streamer should be modified in order to support the alternative model and which advantages we gain by doing so.

#### 7.2.1 Modifications to streamer

As described in section 4.1.2, the streamer can be, and was modified for recording without “much trouble”. The problems appear when trying to implement time-shift since reading and writing to permanent storage changes the nature of the stream flow beyond what the original streamer design can handle. This means rewriting the parts of the streamer that handle the stream flow.

Another part that would need rewriting is the part of the streamer that receives commands from the *Media manager* and controls the elements in the streamer. The reason for this is that a streamer that supports time-shift must behave differently when user tries to pause, fast-forward, ..., or rewind the stream. This is the code module called *streamer core* in figure 3.3 on page 19.
With these modifications, it would be possible to implement seamless\textsuperscript{1} time-shift functionality. Also, a smooth transition from time-shift to live stream would be possible. Stream flow such as depicted in figure 7.1 would then be possible.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{streamer.png}
\caption{Example of how a new streamer could look like. The flow can now be split into two (or several) parts and \textit{streamer core} is modified to support a divided stream flow.}
\end{figure}

\section{Remote storage access protocols}

In chapter 5 on page 36, protocols which can be used for remote file access are discussed. As mentioned earlier, choosing a standard protocol such as \textit{http} is of great interest for standardisation besides the fact that it would

\textsuperscript{1}No frames are dropped when starting/stopping a time-shift, i.e. the stream is played continuously.
make matters simpler for many users. For example, server software might already be installed on a PC, which would eliminate, or make the software installation easier for the end user.

7.3.1 HTTP and UPnP

Using http for remote file access ought to be looked into very carefully, should this solution became an option for the end user. Also UPnP, Universal Plug and Play, deserves a closer look. This is a service discovery protocol which hopefully can be used to discover mass storage devices on the network and negotiate a transfer protocol between them.

However, UPnP is an XML based protocol and the impact of integrating such a protocol into a set-top box streamer must be investigated before any decisions can be made.

7.3.2 RTSP

RTSP was also mentioned as an option in section 5.2.2. This protocol is designed for streaming media so it could most certainly be used for playback of streams from a PC. In current standard there is also a possibility to record a presentation, using the RECORD command [13]. However, RECORD seems to be in danger of getting out of the RTSP standard, which makes the RTSP inappropriate for recording of streams, i.e. transfer of streams from the set-top box to the PC.

7.3.3 RFS

Should the RFS be used in further development, both the client and the server must be extended to support directory listing to have the possibility to browse the server file structure from the client.

Also security, first of all user identification, must be added to both client and the server.
7.4 Recording awareness

Another very important issue in further development is the recording awareness in the system. The most important software modules in need of these modifications are the Media manager and the streamer. Both these modules operate without the knowledge of recording, even though the calls to the recording functions go through their interfaces.

Also call-back functionality between the streamer and the Media manager must be implemented. Even though it is possible to use polling from the Media manager to find out about the changes in the streamers, some kind of call-back functionality would be both easier to use and more efficient in runtime. This functionality can be used for both status and error reporting to the caller.
Appendix A

Glossary

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>Everything displayed to the user, independent of the transfer media or technology. E.g. audio, video, tele text and subtitles.</td>
</tr>
<tr>
<td>DVB</td>
<td>“DVB, short for Digital Video Broadcasting, is a suite of internationally accepted, open standards for digital television maintained by the DVB Project, an industry consortium with more than 300 members, and published by a Joint Technical Committee (JTC) of European Telecommunications Standards Institute (ETSI), European Committee for Electrotechnical Standardization (CENELEC) and European Broadcasting Union (EBU).” [14]</td>
</tr>
<tr>
<td>DVB-T</td>
<td>DVB standard for terrestrial digital TV (tv-over-the-air).</td>
</tr>
</tbody>
</table>

Table A.1: Special abbreviations and expressions used in this document.
<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVR</td>
<td>Digital Video Recorder. See “PVR”.</td>
</tr>
<tr>
<td>ECG</td>
<td>Electronic Content Guide. Guide to provider content, for example accessible via the set-top box. Common ECG information is program start and end, expert comments and short reviews.</td>
</tr>
<tr>
<td>file descriptor</td>
<td>A file handle which allows manipulation of file it is attached to. In unix systems, file handles are also used as handles to all sorts of devices, e.g. the network interfaces.</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol, a protocol used for transferring files on TCP/IP networks. Commonly used on the Internet.</td>
</tr>
<tr>
<td>GNU</td>
<td>“The GNU Project was launched in 1984 to develop a complete UNIX style operating system which is free software.” [15]</td>
</tr>
<tr>
<td>GNU Linux</td>
<td>A free operating system, kernel originally created by Linus Torvalds.</td>
</tr>
<tr>
<td>GPL</td>
<td>GNU Public Licence. Software licence under which a lot of free software is released.</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol. General, clear text protocol used for data transfer on TCP/IP networks, in particular on the Internet.</td>
</tr>
<tr>
<td>LGPL</td>
<td>Free software licence which allows reusing LGPLed software in development of non-free software, without requiring release of this software under the same licence. See also GPL.</td>
</tr>
<tr>
<td>linux</td>
<td>See “GNU Linux”.</td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Picture Expert Group. This is the name for a collection of standards commonly used for encoding digital audio/video.</td>
</tr>
<tr>
<td>MPEG2</td>
<td>MPEG standard for encoding digital video.</td>
</tr>
<tr>
<td>polling</td>
<td>A way of gaining information from an information source by constantly rereading the information with short time intervals.</td>
</tr>
</tbody>
</table>

Table A.2: Special abbreviations and expressions used in this document.
<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS</td>
<td>Network File System. “Network File System (NFS) is a protocol originally developed by Sun Microsystems in 1984 and defined in RFCs 1094, 1813, (3010) and 3530, as a file system which allows a computer to access files over a network as easily as if they were on its local disks.” [14]</td>
</tr>
<tr>
<td>PDR</td>
<td>Personal Digital Recorder. See “PVR”.</td>
</tr>
<tr>
<td>PVR</td>
<td>Personal Video Recorder. Commonly used for describing a system for recording tv shows on digital media.</td>
</tr>
<tr>
<td>RFS</td>
<td>Remote File System. Name of the protocol which was developed during this thesis work for communication between the set-top box and a PC on the local network.</td>
</tr>
<tr>
<td>RFSP</td>
<td>Remote File System Protocol. See “RFS”.</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real Time Streaming Protocol. Protocol used for streaming audio/video on TCP(UDP)/IP networks, such as Internet.</td>
</tr>
<tr>
<td>stream</td>
<td>Sequential audio and/or digital video data. See also transport stream.</td>
</tr>
<tr>
<td>telnet</td>
<td>Remote access protocol, often used on unix systems. In original versions, very insecure since everything, including username and password, is transferred unencrypted.</td>
</tr>
<tr>
<td>time-shift</td>
<td>“Time shifting is the recording of television shows to some storage medium to be viewed at a time convenient to the consumer. “ “A digital PVR also brings new possibilities for time shifting, as it is possible to start watching the recorded show from the beginning even if the recording is not yet complete.” [14]</td>
</tr>
<tr>
<td>transport stream</td>
<td>A term for interleaved streams of, for example, audio, video and subtitling information.</td>
</tr>
</tbody>
</table>

Table A.3: Special abbreviations and expressions used in this document.
<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>See “transport stream”.</td>
</tr>
<tr>
<td>UPnP</td>
<td>Universal Plug and Play. A service discovery protocol based on XML.</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier. “A Uniform Resource Identifier (URI), is an Internet protocol element consisting of a short string of characters that conform to a certain syntax. The string indicates a name or address that can be used to refer to an abstract or physical resource.” [14]</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus. A bus used for attaching external devices to a PC. Version 1 supports a 12Mbit/s maximum transfer rate, while version 2 is much faster at 480Mbit/s. These are theoretical values that might differ from real life experience.</td>
</tr>
<tr>
<td>VoD</td>
<td>Video on demand. “Video on demand systems are systems which allow users to select and watch video content over a network as part of an interactive television system...” [14]</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language. A text based markup language without “special purpose”. Often used to define other markup languages.</td>
</tr>
</tbody>
</table>

Table A.4: Special abbreviations and expressions used in this document.
Bibliography


Copyright

Svenska

Detta dokument hålls tillgängligt på Internet - eller dess framtid omställare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstå.

Tillgäng till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

English

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Aner Gusic
Linköping, 18th March 2005