6. Internationaler Linux-Kongress an der Universität Augsburg

Mit der Unterstützung der Linux-User-Group Augsburg und der Universität Augsburg findet vom 8. bis 10. September 1999 an der alten Universität Augsburg der 6. Internationale Linux-Kongress statt.

Veranstalter sind GUUG, NLUUG, LunetIX und LIVE.

Unterstützt wird die Veranstaltung durch Addison Wesley Verlag, delix, IGEL Technology Labs, iX Magazin, Interest Verlag, Buchhandlung J.F. Lehmanns, Markt&Technik Verlag, O'Reilly Verlag, RedHat, SGI und SuSE.

Programmübersicht

Tutorials

Mittwoch, 8.9.99 12:00-18:00

Am ersten Tag der Veranstaltung besteht die Möglichkeit, von anerkannten Experten für spezielle Themen konzentriertes Wissen und Erfahrung im Rahmen von 6-stündigen Tutorials vermittelt zu bekommen:

1. **Firewall Technologies with Linux as Case Study**
   Referent: Jos Vos

2. **Debian**
   Referent: Roman Hodek and Martin Schulze

3. **Automated installation**
   Referent: Thomas Lange

Konferenzprogramm

Das Programm der 9. und 10. September stattfindenden Konferenz ist in der nebenstehenden Tabelle zusammen-gefasst:

Donnerstag, 9. September 1999

10:00-10:15	Eröffnung			Jens Bachem
10:15-11:00	Keynote: Filesystems 2000			Dan Koren
11:00-11:30	- Pause -			
11:30-12:15	From ext2...	Theodore Y Tso	What's going on at Xfree86	Dirk Hohndel
12:15-13:00	... to ext3	Stephen Tweedie	Frame-buffering with XFree86	Geert Uytterhoeven
13:00-14:30	- Mittagspause -			
14:30-15:15	InterMezzo: a distributed filesystem prototype for Linux	Phil Schwan	Video4Linux	Ralph Metzler
15:15-16:00	Implementation of LVM - a Logical Volume Manager for Linux	Heinz Mauelshagen	ALSA	Jaroslav Kysela
16:00-16:30	- Pause -			
16:30-17:15	XFS on Linux	Jim Mostek	KOffice - an Office Suite for the K Desktop Environment	Reginald Stadlbauer
17:15-18:00	reiserfs	Hans Reiser	**BOF-Sessions**	

Freitag, 10. September 1999

9:30-10:15	Deficencies in the UNIX-Design	Frank Klemm	The making of Linux/ia64	Stephane Eranian
10:15-11:00	Thoughts about init	Lutz Donnerhacke	The Debian Build-Daemon	Roman Hodek
11:00-11:30	- Pause -			
11:30-12:15	Linux Security for Programmers, Crakkers and Managers	Olaf Kirch	MOSIX	Ariel Rosenblatt
12:15-13:00	FreeS/WAN IPsec implementation for Linux	Richard Guy Briggs	MERGEMEM	Philipp Reisner
13:00-14:30	- Mittagspause -			
14:30-15:15	Netfilter: Packet Mangling in 2.4	Paul "Rusty" Russell	WINE	Marcus Meissner
15:15-16:00	NIS and NIS+ for Linux	Thorsten Kukuk	The LinuxDirector VirtualServer Project	Lars Marowsky-Brée
16:00-17:00	Abschlussvortrag			Rishab Aiyer Ghosh

Abstracts zum Linux-Kongress

Im folgenden finden sich die Kurzfassungen der Beiträge zum Linux-Kongress in der Reihenfolge des Konferenzprogramms.

Firewall Technologies with Linux as Case Study

Wednesday, 8.9., 12:00-18:00
Jos Vos

This tutorial presents an overview of generic firewall theory and techniques, as well as an overview of the available implementations of these concepts for the freely available Linux operating system.

The first part covers firewalls in general. It provides necessary background information, explains the most commonly used firewall terminology, and it describes most of the currently known firewall concepts. Furthermore, the various firewall techniques that are available (like packet filtering and proxy servers operating on different levels) are explained and their pro's and con's will be discussed. Some important aspects for designing firewalls are explained using some example firewall architectures.

The second part covers the firewall software available for the Linux operating system. A large number of different software packages for packet filtering and proxy services will be described, as well as auxiliary techniques and software, like network address translation, masquerading, virtual private networks, and various additional tools that can improve the host and network security using Linux systems. This part includes an extensive introduction to ipchains, the Linux 2.2 packet filtering and masquerading software, but it also talks about less well-known software packages, that are often not standard available in most Linux distributions. The tutorial will cover the new techniques and tools available in Linux 2.2, but it also addresses migration from Linux 2.0, as well as future directions.

The Author

Jos Vos is CEO and co-founder of X/OS Experts in Open Systems BV. He has an experience of more than 15 years in the area of research, development, and consulting related to UNIX systems software, Internet, and security. He is the author of ipfwadm and part of the firewall code in the Linux 2.0 kernel. His company X/OS releases its own LinuX/OS distribution, an extension to Red Hat Linux with a lot of additional software packages, often related to security and encryption.

How to make Debian Packages - a brief introduction

Wednesday, 8.9., 12:00-18:
Martin Schulze
Roman Hodek

With Linux entering more and more stages it is often useful to be able to provide additional packages to the distribution one uses. This can improve maintenance, especially when working on a compound product or when keeping software on a park of machines up to date.

Debian GNU/Linux uses a highly sophisticated packaging system. It comes with a well formulated policy describing requirements every package has to meet. Hence, every native Debian package integrates well into the system. As a result Debian machines are easy to maintain and easy to upgrade. This makes Debian a good choice when setting up a machine park.

The Debian packaging system includes maintenance of regular files, documentation, different flavours of programs, kernel, logfiles, cronjobs, menu entries, window-managers, configuration files, configuration tools, configuration interfaces and more.

The workshop will give a brief overview about Debians packaging policy and requirements for packages. It will give explanations how Debian packages look like and will give an introduction into building packages of your own.

This workshop is useful for technicians who want to add functionality to their Debian systems with the goal of providing a compound product. Another field of endeavour would be a department or company using a pile of machines running Debian with the need of adding tools or functionality to their installations. Of course, this workshop is also meant for individuals who want to add local packages, upcoming maintainers for Debian and software developers.

Update #1: Roman Hodek, author of buildd, has submitted a talk about the *Debian Build Daemon*. He will explain how it works wrt. Debian Maintenance at the end of this workshop as well.

Automated installation

Wednesday, 8.9., 12:00-18:00
Thomas Lange

This tutorial presents a technique for a fully automatic installation of a Linux operating system on a PC cluster without any interaction. You can take one or more virgin PCs, turn on the power and after a few minutes Linux is installed, configured and running on the whole cluster.

In the first part I present the parts of an installation and the prerequisites and the preliminary work that has to be done. The creation of a configura-

tion is shown and the technical details are explained.

In the second part the participants can try this technique on available PCs and perform an installation with their own configuration.

Filesystems 2000

Thursday, 9.9., 10:15-11:00
Dan Koren

Modern file systems such SGI's XFS and Veritas' VxFS provide not only outstanding performance and reliability but also a wealth of functionality to support a broad range of commercial applications, such as databases, media servers and hierarchical storage.

Yet more change is to come as we move into the next millenium. New advances in both software and hardware, such as log based file systems, the rise of the Internet, the convergence of network and storage technology and the advent of storage area networks (SANs) are beginning to shape a new generation of file systems that will provide even higher performance and greatly expanded functionality.

This talk will present a brief overview of the state of file system technology, then discuss major trends and attempt to sketch a file system roadmap for the next 10-15 years.

The Author

Dan Koren manages one half of file system development at SGI, including XFS and CXFS for Linux. He has been a system software developer for over 25 years, and has worked on file systems for real time and image processing applications as well as database and transaction processing servers. He was among the developers of the Veritas file system (VxFS) and the Sun Database Accelerator (DBE). Prior to joining SGI he managed Database Engineering at Mips Computers and file system development at Veritas Software.

From ext2...

Thursday, 9.9., 11:30-12:
Theodore Y. Tso

Ted's abstract is not yet available, but he will, together with Stephen Tweedie in the next talk, give us insight about the smooth migration from ext2 to ext3.

What's going on at XFree86

Thursday, 9.9., 11:30-12:15
Dirk Hohndel

This is your typical "past-present-future" talk.

In summer 1999 The XFree86 Project will release the "next generation" of XFree86. This has for a long time been dubbed XFree86-4.0, if this is going to be the official name is not sure, yet. There are tons of changes in that release, many new features, new hardware support, old hardware that is no longer supported, changes to the config file format, a complete new architecture for loading modules at run-time, and so on.

The talk will give a quick overview of what XFree86 is all about, will talk about some of the new features and issues with "XFree86-4.0" and will show some insights into what we are (then) currently working on. Topics that will no doubt be covered include

- changes and new concepts for 2D (multi head, xinerama, overlays)
- ongoing work on 3D
- ideas for configuration

The Author

Dirk Hohndel is Vice President of The XFree86 Project, Inc. and CEO of SuSE Rhein/Main AG. He has worked with and developed Linux since 1991 and has been actively involved in XFree86 since 1992.

... to ext3

Thursday, 9.9., 12:15-13:00
Stephen Tweedie

The talk is the continuation of the previous talk by Ted.

Frame-buffering with XFree86

Thursday, 9.9., 12:15-13:00
Geert Uytterhoeven

The frame buffer device in recent kernels provides an abstraction for frame buffer based graphics hardware.

We will discuss two important aspects of frame buffer devices. Graphical consoles are implemented in the kernel on top of the frame buffer device. A second topic is the access of the graphics hardware by user applications. The most important application here is the (accelerated) X server.

InterMezzo: a distributed filesystem prototype for Linux

Thursday, 9.9., 14:30-15:15
Phil Schwan
Peter J. Braam
Michael Callahan

Advances in clustering, failover systems, and distributed data storage are some of the important developments in storage software technology made during the last several years. Although decreasing hardware prices

have made large storage subsystems available to a much wider audience, the cost of software with these advanced features has remained prohibitive. Commercial filesystems such as AFS and those sold by Veritas remain too costly for individual users, and free projects like Coda--though impressive--are not ready for production use. The goal of the InterMezzo project is to provide the features and techniques developed by AFS and Coda, but to do so symmetrically, asynchronously, and an order of magnitude more simply.

The core of InterMezzo is a Linux kernel module named Presto. Presto acts as a filter driver for application VFS requests; it pre- and post-processes requests, fulfilling cache misses and maintaining a log of updates which are lazily shipped to the server. Presto also receives VFS requests for operations made by the userlevel cache manager; these requests are passed directly to the underlying filesystem. In essence, InterMezzo is layered on top of a normal disk filesystem (such as ext2), and therefore enjoys its robustness, performance, and scalability. Furthermore, unlike the Arla and Coda projects, we avoid the burden of writing and maintaining our own disk filesystem. By keeping the kernel module simple and small, we minimize both overhead and the possibility of bugs.

The cache manager and file server is known as Lento. It uses Perl and the Perl Object Environment (POE) package to implement an asynchronous framework; our mechanisms favour asynchronous request processing over multiple threads of control for better management of the complicated state of the file server and cache manager. Processing multiple outstanding requests from the kernel and from TCP sockets and parallelising outgoing RPCs to multiple systems is done using Perl object classes. The cache manager and file server enforces security, negotiates version stamps and callbacks to avoid seeing stale data, and handles write permits to try to avoid write conflicts. Furthermore, all of the semantic decisions that the file server makes during these processes are cleanly separated from the rest of the code, allowing one to customize InterMezzo behaviour for a certain task or environment.

Careful examination of Coda's callback and reintegration protocols revealed an opportunity to implement Lento with an extreme degree of symmetry and code re-use by allowing it to act as both a client and server. A Lento client downloads directories and files from the server, while the server uses the same mechanisms to download journalled updates from the client. This code reuse helps keep Lento small, and makes it easier to understand and modify.

InterMezzo is fundamentally different from Coda in using a write-back cache, avoiding synchronous interactions with the server. This allows operations to proceed at nearly the speed of the local disk filesystem, while journalled modification logs can be optimised and lazily shipped to the server in the background. This is clearly a big win in terms of performance.

The InterMezzo code base (~2,500 lines of C for Presto, ~5,000 lines of Perl for Lento) is small enough to provide an attractive research platform. Future plans include features such as server replication and disconnected client operation that Coda currently supports.

The Authors:

Peter J. Braam <braam@intermezzo.org is a Senior Systems Scientist at CMU where he is leading the Coda Project. Peter's interests are in distributed file and storage systems and he has worked on a variety of projects in this field, almost always involving Linux as the primary platform. Peter received his PhD from Oxford University in 1987 and subsequently held faculty positions at the Unversity of Utah and at Oxford, before joining CMU in 1996. Peter is also President of Stelias Computing Inc., which specializes in consultancy on Linux file and storage systems. When not at work, he can perhaps be found in his Alaska cabin, or on a nearby mountain.

Michael Callahan <mjc@intermezzo.org is currently Director of Advanced Development at Ask Jeeves, Inc. Previously he has worked on networking and distributed filesystems, scientific visualization, and digital micropayments. He was trained as a mathematician at Harvard and Oxford Universities.

Phil Schwan <pschwan@thepuffingroup.com is a long-time Linux hacker with a recently-found fondness for filesystems. He began his undergraduate study at Carnegie Mellon University, and currently does filesystems hackery as a contractor at Seagate Technologies, Inc. Past projects include a highly Linux-optimised FTP server (hFTPd), LCD panel driver work, and small GNOME dabblings.

Video4Linux

14:30-15:15 (Hörsaal II)
Ralph Metzler

In the last few years Linux has established itself as a very successful alternative to conventional operating systems as a fast platform for file and web servers. This can be attributed to the incredibly fast rate of development and improvement of the networking and file system support code in the Linux kernel. But support for multimedia hardware is still far behind that for other operating system, sometimes due to the lack of support and documentation by hardware manufacturers, sometimes due to licensing problems (most notably with

DVD). Another major reason for this was the lack of a standardized programming interface. Although from early on there were some Linux drivers for multimedia cards available they usually came with their own set of ioctls and their own application which would only work with this one card.

In 1997, Alan Cox introduced the Video4Linux API to the 2.1.x series of kernels. It acts as a layer between a low level device driver and the user application. The basic idea is that once the device driver registered itself with the Video4Linux layer it can be used by the user application via the Video4Linux API ioctls. Thus, any application which is fully conforming to the Video4Linux standard (I admit that not all do this FULLY) should be able to use any Video4Linux conforming device driver.

In the beginning, Video4Linux only provided support for bttv and the Quickcam parallel port cameras. Since then it has come a long way. In the latest 2.3.x kernels the support ranges from radio cards, over TV cards to MJPEG cards. Other separately available drivers which use the Video4Linux standard (in a slightly modified form) even include an MPEG2 decoder card. But Video4Linux also still has to go a long way to become a full-fletched Video API which can support all kinds of multimedia hardware. The current version is e.g. not very flexible regarding capturing of video streams and is still very much influenced by the specific features of the bttv driver. The next version of the API, Video4Linux2, is much more flexible and was also designed with the support of new hardware like MPEG encoders in mind.

My talk will be roughly structured like this:

- What is Video4Linux? (Motivation and historical development.)
- Structure and components of a Video4Linux driver
    - the Video4Linux API
    - Video4Linux implementation in the kernel and driver registration
    - general supporting modules: I2C, tuner, TV encoder, sound decoder, ...
- Current and future developments
    - new features of the bttv driver
    - Video4Linux2
    - support for new hardware: MPEG2 (DVD and DVB), MJPEG

Implementation of LVM - a Logical Volume Manager for Linux

Thursday, 9.9., 15:15-16:00
Heinz Mauelshagen

A Logical Volume Manager (LVM) is a subsystem for on-line disk storage management which has become a de-facto standard accross UNIX implementations.

It adds an additional layer between the physical peripherals and the i/o interface in the kernel to get a logical view of disks. Unlike current partition schemes where disks are divided into fixed-sized sections, LVM allows the user to consider disks, also known as physical volumes (PV), as a pool (or volume) of data storage, consisting of equal-sized extents.

A LVM system consists of arbitrary groups of physical volumes, organized into volume groups (VG). A volume group can consist of one or more physical volumes. There can be more than one volume group in the system. Once created, the volume group, and not the disk, is the basic unit of data storage (think of it as a virtual disk consisting of one or more physical disks).

The pool of disk space that is represented by a volume group can be apportioned into virtual partitions, called logical volumes (LV) of various sizes. A logical volume can span a number of physical volumes or represent only a portion of one physical volume. The size of a logical volume is determined by its number of extents. Once created, logical volumes can be used like regular disk partitions - to create a file system or as a swap device.

The talk includes a demonstration of basic usage szenarios of the LVM including the initialization of physical volumes and setup of volume groups. Logical volumes will be created and resized. Extents will be moved between phyiscal volumes to show the non interrupting way this happens. An illustration of the basic metadata structures, the implementation of the driver, some essential tools and library functions finishes the talk.

ALSA

Thursday, 9.9., 15:15-16:
Jaroslav Kysela

Introduction

The Advanced Linux Sound Architecture (ALSA) project is originated from the Linux Ultra Sound Project. The original project was intended only for Gravis UltraSound soundcards. The ALSA project extents this effort for all soundcards and sound APIs.

The main goals of the ALSA project are:

- Create a fully GPL sound driver system for Linux.
- Create a fully modularized sound driver.
- Maintain backwards compatibility with most OSS/Free applications.
- Create the LGPL ALSA Library (C, C++), which simplifies ALSA application development.
- Create the ALSA Manager, an interactive configuration program for the driver.

Soundcard and devices

The ALSA driver uses a quite different look to the devices than the OSS/Free driver. The primary identification element is a soundcard. The devices for one interface - like PCM - are not mixed together. Each soundcard has its own set of devices. The ALSA driver identifies devices via two numbers: a soundcard number and a device number. This representation is more real and an user understands the device mapping in an easier way.

Modularized design

The ALSA driver is very modularized. We are trying to separate every things to independent modules. This separation has big advantages:

- The code can be reused. The most of current soundcards have same components.
- The midlevel code separation from the lowlevel drivers causes that the lowlevel drivers are smaller and more readable. It allows a better maintaince.

Current status of native interfaces in the driver

- **Control**
  The control interface allows to an application to obtain various information from the driver without the exclusive locking of some feature. It also offers the reading of the change/modify/add/remove events for the universal switches.
- **Mixer**
  The mixer (version 2) interface has very new design now. It does not follow anything known. The mixer is described via control elements and routes among them. It allows to describe very complicated audio analog/digital mixers available in current soundcards. We have also possibility to enhance this interface in each aspects. The mixer interface has also a group control for the basic elements. This control method was designed to provide the standard mixer behaviour and to retain the compatibility with the OSS/Free mixer interface. Both control methods (element & group) can be used concurrently. The mixer interface also fully supports the change/modify/add/remove notifications through events which can be obtained via the read() function. It allows a perfect synchronization among more mixer applications.
- **PCM (digital audio)**
  The PCM interface is probably most stable at this moment. It supports full duplex when hardware contains this feature. The last added feature is multi open. When hardware is able to mix more PCM streams concurrently, then the ALSA driver allows the open call more times, until the resources are not exhausted.
- **Raw MIDI**
  The Raw MIDI interface is fully supported. It allows to read and write unchanged data from the MIDI ports. These ports can connect either external or internal devices.
- **Timer**
  The timer interface allows to use timers from soundcards. It also offers a slave mode. It means that some piece of code from the user space can be synchronized with any kernel device which uses same timer. The PCM stream can be also used as a primary timer source. It allows a perfect synchronization between a digital audio stream and a MIDI stream.
- **Sequencer**
  The sequencer inside the ALSA driver is probably the most interesting part of the ALSA driver, but it is still under development. The original proposal from Frank van de Pol have been very enhanced and the current code is able to drive 192 clients at same time. Each client has two memory pools for input and output events. The features like event merging, a client notification about the internal changes or changes provoked with an another application are a matter of course.
- **Hardware dependent**
  This interface does not pro-

vide an abstract layer to applications except the protocol identification. It is intended for hardware which does not fit to any abstract interface like FX processors, a special access to synthesizer chips etc.

Current status of OSS compatible interfaces in the driver

- **Mixer**
  The OSS/Free mixer API is fully emulated.

- **PCM (digital audio)**
  The OSS/Free PCM API is emulated. A few applications have trouble with this emulation. These trouble are mostly caused with wrong assumptions of authors.

- **Raw MIDI**
  The OSS/Free raw MIDI API is emulated.

- **Sequencer**
  The level 1 sequencer API is emulated. The level 2 API is not emulated. There is available only a few application for this interface.

Related code

The ALSA driver also contains the full support for ISA Plug & Play devices. The new isapnp module is not designed only for soundcards, but it can be used with an another driver for an ISA Plug & Play device. The only problem may be the kernel merge.

A next part of the ALSA driver is the persist module which allows to store and restore run-time information. The ALSA driver uses this module for better kmod support.

The C ALSA library

The ALSA C library fully covers the ALSA driver API. The ALSA library extents the driver functionality and hides the kernel calls.

The future

The interfaces are closely before completion. We are hardly working on the ALSA native sequencer in the driver and drivers for wavetable chips. We also need to write the documentation for application and lowlevel driver programmers.

The ALSA team hopes that the ALSA driver will be merged into 2.3 kernel tree. Alan Cox partly confirmed this fact. We expect that OSS/Free and ALSA drivers will coexist together for a while. OSS/Free will be removed from the kernel tree after some time when ALSA will contain all lowlevel drivers from OSS/Free.

About the Author

Jaroslav Kysela is the founder and current leader of the ALSA project. He has been working with Linux since 1993. Between 1994 and 1997 he worked on the Linux Ultra Sound Project. In 1995 he initiated and still maintains the development of the driver for 100Mbit/s Voice Any Lan network adapters from Hewlett Packard (the hp100.c driver). Since 1998 he has been working on the ALSA project. He was hired by SuSE GmbH in April 1999.

WWW links

Advanced Linux Sound Architecture - http://www.alsa-project.org
The Linux Ultra Sound Project - http://www.perex.cz/~perex/ultra
Open Sound System (OSS) Free - http://www.linux.org.uk/OSS
Open Sound System (OSS) Programmer's Guide - http://www.opensound.com/pguide
SuSE - http://www.suse.com

XFS on Linux

Thursday, 9.9., 16:30-17:15
Jim Mostek

SGI is porting is world class XFS file system to Linux. XFS is a full 64-bit file systems that can scale to handle extremely large files and file systems. Some XFS features include:

- journaling for very fast recovery (no fsck),
- delayed write allocation,
- efficient use of the buffer and/or page cache, and
- direct I/O.

This talk will describe the architecture and implementation of XFS and discuss the Linux porting effort.

About the Author

Jim Mostek is the technical lead of the XFS Linux port at SGI. Jim has a Math and Computer Science degree from the University of Illinois. He has over 18 years of extensive experience in file systems and networking including:

- key CXFS developer,
- extensive internals experience with XFS, DFS, threads,
- File System Switch (FSS), VFS/Vnodes, UNIX streams, and sockets.

Jim's primary experience has been with UNIX based systems of various

flavors including BSD, PWB, and SystemV, IRIX, and UNICOS.

KOffice - an Office Suite for the K Desktop Environment

Thursday, 9.9., 16:30-17:15
Reginald Stadlbauer

With desktop environemnts like KDE and GNOME these days Linux becomes more and more popular as desktop system too. But a nice desktop without applications is quite useless. So desktop applications are urgently needed. The KOffice will provide such a set of applications.

The desktop applications, which are mostly used (like wordprocessors, spreadsheets, presentation Applications, etc.) are normally bundled together to an Office Suite. Today companies like Stardivison, Corel and ApplixWare offer their Office applications for Linux too. Some of them even provide a private edition which everybody can download without paying for it. So, why do we need another Office Suite like the KOffice?

There are some good reasons. First of all, the Office Suites I mentioned above, are not free. This means the sources are closed. In contrast to that the KOffice and all its sources are free - it's under the GNU General Public License. Besides the advantage that everybody may change and redistribute the sources, people which are working on not so popular systems (like Alpha Linux, Sparc Linux, etc.) can just compile the sources on that system too and don't need to wait until a company maybe supports the certain hardware platform.

But there is one more important reason. If we look at the existing Linux Office Suites, we see that they don't cooperate with others. So it's e.g. not possible to use the spreadsheet of one Office Suite in the wordprocessor of another. It's most of the time not even possible to exchange data between different Office Suites via drag'n'drop.

So the KOffice team developed a free object model (KOM/OpenParts) based on the industry standard CORBA. Using this object model you cannot only embed KOffice applications into others but also other applications, which use the OpenParts object model, will be able to cooperate with KOffice and other OpenParts applications. E.g. the whole KDE 2.0 desktop is built upon this object model to make it as flexible as possible. If one day somebody writes a better chart engine than KDigramm using the OpenParts model, it's e.g. possible to visualize the data of KSpread with this new digramm engine.

But which applications are already included in the KOffice? There is KSpread, an extensible spreadsheet application, which was also the first KOffice program. KWord is the wordprocessor of the KOffice. In contrast to most of the availabe wordprocessors, it's framebased like FrameMaker and Ami Word. But it's also possible to use it page orientated and for simple tasks like writing letters. The presentation application of the KOffice, called KPresenter, allows to create screenpresentations, printed presentations and HTML slideshows. There is also a quite advanced vector drwing program, named KIllustrator, included in the KDE Office Suite. To be able to embed pictures into applications, which don't support that (like KSpread), there is an image viewer, which also offers simple image manipulation functions, called KImage. KDiagramm, which is mostly used as embedded part too, allows to visualize data with lots of different chart types. It also works perfectly together with KSpread. Then there is the formula editor KFormula, which also is most useful as an embedded part. The youngest member of the KOffice is Katabase, a database application, which is in a very early development state.

One feature, that made Office Suites like the Microsoft Office very successful and popular is scripting, which makes the Office applications extensible (in this case VBA). So originally the KOffice team started to make the KOffice scriptable with Python. But after some problems with Python, we decided to write a new scripting language, called KScript. The syntax is very much like Python, but as KScript is designed for the KOffice and the OpenParts object model, it's easier to integrate this into the KDE Office Suite. So it's already possible to extend KSpread using KScript - the other KOffice components will follow.

Another very important thing for exchanging data between different applications and Office Suites are file filters. So the KOffice provides a very flexible filter architecture - all filters are standalone applications. A filter gets the data of a file, the mime type of that file and the mime type of the file, which the filter should return. But a nice architecture without working filters doesn't help a lot. So there are already ASCII and HTML import and export filters for KSpread and KWord, and KIllustrator can read XFIG files. Also people are working on a RTF and a WinWord filter for KWord and a MIF filter (FrameMaker) for KWord has been started too. As native file format all KOffice applications use XML, which allows to edit KOffice documents in a texteditor too.

As nobody is able to write Office applications in a very short time, such a development takes a lot of time. Version 1.0 of the KOffice will be distributed together with KDE 2.0. A first beta may be expected in autumn 1999.

reiserfs

Thursday, 9.9., 17:15-18:00
Hans Reiser

Reiserfs is a file system using a variant on classical balanced tree algo-

rithms. The results when compared to the ext2fs conventional block allocation based file system running under the same operating system and employing the same buffering code suggest that these algorithms are more effective for large files and small files not near node size in time performance, become less effective in time performance and more significantly effective in space performance as one approaches files close to the node size, and become markedly more effective in both space and time as file size decreases substantially below node size (4k), reaching order of magnitude advantages for file sizes of 100bytes. The improvement in small file space and time performance suggests that we may now revisit a common OS design assumption that one should aggregate small objects using layers above the file system layer. Being more effective at small files DOES NOT make us less effective for other files, this is a general purpose FS.

Hans Reiser will also discuss whether filesystems are semantically impoverished compared to database and keyword systems, and whether that should be changed.

Deficencies in the UNIX-Design

Friday, 10.9., 9:30-10:15 (Hörsaal I)
Frank Klemm

It is difficult to use and to program for the Unix enviroment. This lecture tries to find out why. There are three areas:

- kernel space
- system libraries
- user space applications

Indeeds you can find huge deficencies in all three areas. The lectures points out these in the kernel space.

kernel space:

- console/HID support
- removable media support
- modern file system
- distributed devices
- process management
- Checkpointing
- Real time support
- guaranteed I/O rates

About the Author

Frank Klemm studied physics at the university of Jena. Currently he is a DSP programmer at Carl Zeiss Jena. He has worked with Linux since 1996.

The making of Linux/ia64

Friday, 10.9., 9:30-10:15 (Hörsaal II)
Stephane Eranian

Since February 1998, HPLabs has been working to port Linux to the IA-64[1] architecture, an activity which is now part of a broader industry effort. The goal of this project is to build a fully functional, fully optimized Linux distribution and jump-start a Linux community around this new platform. As soon as machines become generally available sometime in 2000, we intend to contribute our source code to the open source community and eventually get it integrated in the official Linux code base. This paper describes our effort so far.

IA-64 is more than just another architecture. It introduces a new computing paradigm called EPIC (Explicitly Parallel Instruction Computing) which means that it exposes instruction level parallelism to the user. This new architecture has massive amount of resources with 128 general registers as well as 128 floating point registers. It also provides very advanced features like instruction bundling, a register stack engine (RSE) to avoid saving registers on procedure calls, predication to minimize the number of branches, speculation to hide memory access latency and rotating registers for software pipelining. In the full paper, we intend to give simple examples with actual code sequence for each feature.

Porting the Linux system to a new architecture involves significantly more than just getting a kernel. A complete port requires a development tool chain, a kernel and thousands of user level applications and libraries. The lack of near-term IA-64 hardware platform make this port even more challenging as we're required to use a simulation environment.

The first step was to put together the development tools. Linux and especially its kernel are heavily dependent on a GNU C compatible compiler. Such tool did not exist, so we decided to work on it first. The obvious candidate was `egcs`, a very active branch off the `gcc` development tree. We also ported the `gas` package to get an assembler, linker and associated `binutils`. Writing an optimizing compiler for EPIC is not a trivial task and would have required changes to the `egcs` front-end and, as such, was out of the scope of our project. Instead, we built a back-end for `egcs` that would produce functional code for now. The tool chain uses the standard ELF64 defined for IA-64 as the object file format and follows the LP64 (longs and pointers are 8 bytes) programming model. By September, the tool chain was able to pass the `egcs` test suite and generate the classic "`Hello World`" program correctly. In the final paper we intend to give more details about this first phase of our work.

We are using an instruction set simulator for execution of IA-64 code. This means that we only simulate the full IA-64 instruction set of the CPU and not the whole platform (e.g. no PCI bus nor BIOS). The simulator provides hooks to get I/O services from the host system. It supports two modes of simulation: user-mode or system mode. The former allows to run user-level applications and it traps into the simulator for system call emulation. In system mode, kernel bring up is possible as the full VM and interrupt behaviors are simulated. We'll describe the work we've done to host this simulator on Linux, and how we've emulated devices like a disk, serial console and Ethernet.

Once the tool chain and simulator were in place, the full development environment was available on Linux/x86 and work on the kernel could really begin.

We decided to do a straight port of the Linux kernel and minimize modifications to the machine independent part.We also chose to keep track of the latest kernel versions (we started with `v2.1.126` and are now using `v2.3.X`) as this would make the final integration stage much smoother. The kernel is running in native 64bit mode and uses little-endian byte ordering for obvious compatibility reasons with x86. The default page size is 8KB (may evolve as we move forward).

We took the incremental approach of bringing up subsystems one by one. We began with `start_kernel()` and then added missing pieces as they became required. We started in late October and ever since we have been enabling key components like VM, interrupts, context switches, system calls,etc. We developed a series of simulated device drivers which would trap into the simulator to get I/O access. We built a SCSI driver (`simscsi`), a serial driver (`simserial`) and an Ethernet driver (`simeth`) which we'll describe in more details in the final paper. By January 1999, we were able to execute our first "`Hello world`" program in user-mode. At that time, we did not have a full C library so recompiling existing commands was not possible. Instead, we had a "&micro" which we used to rewrite simple tools like a tiny shell (`tsh`) and commands like `ls`, `cp`, `cat`, `mount`, `halt`, etc. In early March, the network stack was up and running and we could ping in & out or remote login into the simulated kernel. Shortly thereafter, the signal handling and `ptrace()` support were also in place. We even got `strace` working, which helped with debugging. Today, most of the key components exist and the kernel is complete enough that we can now run actual Linux commands from standard distributions. In the full paper, we intend to give more details of the kernel work and plan on showing actual code sequences that exhibits some of the IA-64 features.

In the meantime, CERN[2] decided to join this effort and started working on the C library. Here again, we selected the latest version (`v2.1`) of the GNU C library as the candidate. The goal was to generate a straight port first without any optimizations so that we could start using it quickly and then later add EPIC-style optimizations to both C and math libraries. After a few weeks, we got a first code drop which allowed us to start recompiling real world applications.

We quickly managed to rebuild a complete login sequence which typically includes `init`, `mingetty`, `login`,`passwd` and a shell. We used the standard RPM[3] packages with some tweaking in the Makefiles to get them to work in a cross-compilation environment. Soon, we had the other basic packages, like `util-linux`, `sh-utils`, `fileutils`, `netkit-base`; some were still incomplete because of missing libraries, like `curses`. We even got shells like `pdksh` and `bash` to run (although not totally functional yet) and also `vim`, a vi-clone.

As more applications were being ported, the usability of our development environment and system mode simulation speed became a real concern. User applications don't necessarily require to run on the actual IA64-kernel, so user-mode emulation is often just fine. Operating in a cross compilation environment was annoying because most standard RPM packages are not really constructed to be recompiled that way. Fixing the Makefiles can be really tedious as some applications use helper programs during the build phase. For those reasons, we decided to work on getting what we called "NUE" (Native User Environment) which would give the illusion of a native IA-64 environment on our x86 host. This was very simple to achieve when combining our simulator (user-level mode) and Linux's `binfmt_misc` code inside a chroot'ed tree. Now you can invoke standard x86 and IA-64 binaries directly from your shell prompt. In this closed environment, `as`, `cc`, `ld`, really are cross compilation tools but look like native. The C library and include files are in their natural location. Thus, it becomes very easy to rebuild RPM packages without changing one line of Makefile: a simple `rpm --rebuild` suffices most of the time.

As of today, we have a complete development environment and we can run either on top of the Linux/ia64 kernel or in the NUE environment. Most of the kernel is done but SMP, platform specific code, the bootloader, IA-32 emulation are still missing. At the user-level, the C library still needs more work and dynamic linking is in progress. On the application front, required packages like `gdb`, `Xfree` server, `Java`, `Mozilla`, `GNOME/KDE` need to be ported. Lots of opportunities for EPIC-style optimizations are available in assembly code but also in C code and as the tool chain improves we hope too

see major performance boost. We're also collaborating with other industry partners like Cygnus, Intel, SGI and VA Linux Systems to make sure we get the best possible Linux system available from day one.

[1] See http://www.hp.com/go/ia64

[2] Centre Européen de la Recherche Nucléaire -- Geneva, Switzerland, see http://www.cern.ch

[3] RedHat Package Manager, see http://www.rpm.org

Thoughts about init

Friday, 10.9., 10:15-11:00
Lutz Donnerhacke

INIT(8) - Linux System Administrator's Manual

NAME

init - process control initialization

SYNOPSIS

/sbin/init {optons}[[action]service{ops}]

DESCRIPTION

Init is the father of all processes. Its primary role is to create processes from a script stored in the file /etc/init.conf (see init.conf(5)). This file usually has entries which cause init to spawn getys on each line that users can log in. It also controls autonomous processes required by any particular system.

Due to the fact all damons run with init as parent, only init is notified when a daemon dies. So init ist the natural process to handle all events of daemons, like starting, reloading, respawning etc. pp..

As a consequence /var/run/ is obsolete. /var/run/utmp and /var/log/wtmp are considered obsolete, too.

SERVICES

A service is a software configuration of the system. Each service may depend on other services running. So a dependency tree is spawned to keep the system running.

Each service can be up or down (or on the way to reach such a final state). The service starts with an call section, waits for all needs to come up and execute the start section to come up itself. The stop section is executed to bring the service down. If a action on starting fails (return anything beside 0), the service will go down.

Daemons can be started by remembering the process ID in a variable. If such a daemons dies, an approbriate dead section is executed. Of course the variable is available to other commands i.e. to send the process a signal.

The reload section is only executed on user's request. If no reload section is defined, the service is stopped and restarted.

BOOTING

After init is invoked as the last step of the kernel boot sequence, it looks for the file /etc/init.conf and tries to parse it. If this fails, init stops.

The first service in /etc/init.conf is the startup default. If an argument was given to init, the named service is started instead of this default.

CHANGING SERVICES

xxx

ENVIRONMENT

Init sets the following environment variables for all its children:

INIT_VERSION
As the name says. Must not used to determine if a script runs directly from init. Use getppid(2) instead.

INTERFACE

xxx

SIGNALS

Init reacts to several signals: xxx

FILES

/etc/init.conf

WARNINGS

Init assumes that daemons do not try to fork to detach. There is no need for such behavior.

AUTHOR

Lutz Donnerhacke <Lutz.Donnerhacke@Jena.Thur.De> due to a request of Frank Klemm <pfk@uni-jena.de>.

The Debian Build-Daemon

Friday, 10.9., 10:15-11:00
Roman Hodek

Introduction

The Debian distribution now supports seven different architectures, but the package maintainers usually only compile binary versions for i386. Developers for other architectures have to watch out for new versions of packages and recompile them if they want to stay up-to-date with the Intel distribution.

As Debian/m68k (the first non-Intel port) started, all this was done manually: One watches the upload mailing list for new packages and takes some of them at home for building. Coordination that no package is built twice by different people was done by announcing on a mailing list. It's obvious that this procedure is error-prone and time-consuming. This has

been the usual way for keeping non-i386 distributions current for a long time.

The build daemon system automates most of this process. It consists of a set of scripts (written in Perl and Python) that have evolved over time to help porters with various tasks. They have finally developed into a system that is able to keep non-i386 Debian distributions up-to-date nearly automatically.

Overview

The `buildd` system consists of the following main:

wanna-build

gives access to a central database (for one architecture) that stores package states, versions, and some other infos.

buildd

queries the database for packages that need rebuilding, takes care of their compilation and uploads them after the build log has been acknowledged by the administrator.

sbuild

is responsible for the actual compilation of packages. It mainly uses standard Debian tools for this, but also takes care of source dependencies and some other minor quirks.

quinn-diff

is used to feed the wanna-build database with new packages. It compares the available package versions for two architectures and outputs the differences.

andrea

generates some source dependencies automatically and merges these data with dependencies added manually.

Outline of Operation

At the heart of the system is the `wanna-build` database, which keeps track of package versions and states. `quinn-diff` compares the package lists for i386 and the target architecture every day and feeds a list of packages that need recompilation into the database where they enter state `Needs-Build`.

All the build daemons (there can be more than one) query the database regularly for such packages and take some of them so that they go into state `Building`. Of course, humans also can take packages, e.g. in special cases where automatic compilation isn't possible. Here we also see the second purpose of `wanna-build`: It ensures that the same version of a package won't be built twice.

If everything goes well, a finished package can be uploaded later, which is another state `Uploaded`. After that it will eventually be installed into the Debian archive so it appears in the updated package list for the target architecture. This list will be merged into the database, so the package will go to state `Installed` and remains there until the next i386 version.

There are several other states; they include: `Failed` is for packages that failed to build due to errors in the sources, and the errors are expected to be fixed in a successor version (after reporting the problem, of course). So a new version will directly enter `Needs-Build`, but with a warning that something was wrong with the previous version. Along with this state an error description is stored. State `Dep-Wait` is used when a package needs some other packages to be compiled but those aren't available yet and must be built before. This state stores a list of required packages and maybe versions, and if all of them are known to be installed the state changes back to `Needs-Build`.

As we have already seen, the build daemon takes packages from the database for compiling them. Let's look a bit closer: If it has some packages to build, it uses `sbuild` for the actual compilation process, and for each build a log is mailed to the maintainer of the daemon. He reviews the log and decides what to do with the package: upload it, set it to `Failed` or `Dep-Wait`, make some additions to the source dependency list and retry it, etc... If a positive acknowledge is received, the daemon moves it to an upload directory, from where all packages are uploaded by a cron job.

Looking at the log files is the only human intervention in the whole process if no errors happen. There are two good reasons for not further automating this: First, sometimes builds end with an ``OK'' result but the build nevertheless failed for reasons that are invisible to the machine. And second, directly uploading would require to automatically PGP-sign the resulting files with a key without passphrase on the build machine. I considered this an unacceptable security hole.

The build script `sbuild` more or less just calls some standard Debian tools to compile the sources. It also helps with some common tasks and bookkeeping, but the really special thing about it are the source dependencies. Often packages need other packages installed for compilation, for example compilers and libraries. It is not practical to have all these packages installed all the time, and often it's not even possible because of conflicts. The source dependencies now simply tell `sbuild` for each package which other packages are needed. It can then automatically install them before the build and remove them again afterwards.

The source dependency list can partially be generated automatically, too, by looking at the dependencies of the binary packages generated by the source. This is `andrea`'s job, which analyses the i386 package list for dependencies and maps library packages to development package names. It also merges the results with manual additions for things that can not be auto-generated, like compilers or special tools.

Applications and Future

At the moment four Debian architectures use `buildd`: m68k, powerpc, alpha and sparc. m68k has even two daemons running, one for the stable release level and one for the rest. And there are plans for a `buildd` on sparc64 (UltraSparc).

The success of `buildd` in keeping a distribution up-to-date most dramatically shows for m68k: This architecture has the slowest machines, but only one daemon can keep pace with the i386 unstable distribution by minimizing idle times and time-consuming manual actions. In the past, we usually always were 200-300 packages behind i386, but with the daemon this has decreased to less than 5% of all packages, including failed and waiting packages.

Another application of `buildd` could be the maintenance of distribution variants, like a Pentium-optimized version. This can be seen as a new ``architecture''. It also would be no problem to, e.g., recompile only packages of a certain importance.

Another proposal is to run a `buildd` for i386. This makes sense, as Debian now has some maintainers that have no i386 machine. This, however, requires some changes first, because i386 is still considered the reference architecture inside the system.

Credits

Most parts of the `buildd` system have been written by James Troup and me. I also want to thank the other members of the Debian/m68k build team (some, but not all: Chris Lawrence, Michael Schmitz, Christian Steigies) and PowerPC porter Hartmut Koptein for suggestions, ideas, and testing. Special thanks go to Stefan Gybas for donating the 68060 accelerator board for kullervo, and last but not least thanks to Martin ``Joey'' Schulze for hosting two of the `buildd` machines, kullervo (m68k) and tervola (PPC).

Linux Security for Programmers, Crackers and Managers

Friday, 10.9., 11:30-12:15
Olaf Kirch

For quite some time, the sendmail mail transport system was the epitome of bad security -- every couple of months, somebody would find a bug or misfeature what allowed local and occasionally even remote attackers to obtain root privilege. Then, Java came along and Eric Allman was off the hook for a while. Currently, the spotlight is on Microsoft because they're new to the multiuser/network game, and their large installed base make them an interesting target.

Tomorrow, it might be Linux.

I have been actively involved in Linux security for at least four years now; both in the role of blundering programmer, as someone catching other people's blunders, and on various mailing lists. Linux' track record is no better or worse than most of the other operating systems, but there's definitely room for improvement.

The amazing news is that many programming mistakes that can lead to security problems seem to be impossible to eradicate. For instance, it has been a fairly well known fact for quite some time that special care needs to be taken when executing another program from a setuid context. However, there are still programs being written today that do not take the proper precautions. The last case I came across was in May 1999, and I'm sure it won't be the last.

There are probably many reasons why things are like this. Speaking from my own experience as a blundering programmer, we tend to develop a partial blindness when it comes to spotting trouble areas in our own code. We're so much in love with our elegant design that we're reluctant to anticipate all the ways in which an attacker might try to break it. In addition, most security pitfalls -- the notable exception being buffer overflows -- look harmless enough until you emph{know} how an attacker might exploit them. Finally, many people underestimate the cleverness of the cracker community. To most people, a cracker is a freaked-out 16 year old kid running scrounged up exploit programs they don't even half understand. However, there's a non-negligible proportion of creative hakker crackers who are well-educated in Unix (in many cases thanks to Linux), and quite clever when it comes to finding the weak spots in your software.

The focus of my talk will be on traps we tend to fall into, and how to avoid them.

Author info:

Olaf Kirch has been a member of Linux community since the 0.97 days. He wrote the Linux Network Administrator's Guide, maintained the NFS code for quite some time, and is currently employed by Caldera.

MOSIX

Friday, 10.9., 11:30-12:15
Ariel Rosenblatt

Mosix is a software tool for supporting cluster computing. It consists of kernel-level, adaptive resource sharing algorithms that are geared for high performance, overhead-free scalability and ease-of-use of a scalable computing cluster. The core of the Mosix technology is the capability of multiple workstations and servers (nodes) to work cooperatively as if part of a single system.

The algorithms of Mosix are designed to respond to variations in the resource usage among the nodes by migrating processes from one node to another, preemptively and transparently, for load-balancing and to prevent memory depletion at any node. Mosix is scalable and it attempts to improve the overall performance by dynamic distribution and redistribution of the workload and the resources among the nodes of a computing-cluster of any size. Mosix conveniently supports a multi-user time-sharing environment for the execution of both sequential and parallel tasks.

So far Mosix was developed 7 times, for different version of Unix, BSD and most recently for Linux.

FreeS/WAN IPsec implementation for Linux

Friday, 10.9., 12:15-13:00
Richard Guy Briggs

Why be concerned with network security? The internet was created as a government research network where it was assumed that no one was hostile. It migrated to universities and other research institutions that were not generally accessible to the public. Where it ran through hostile areas, the hardware and physical carriers were secured. While students started to gain access to such systems, it sufficed to secure the exposed terminals and user interfaces.

Since then, it has exploded in use because the public started to gain access to its resources: databases, bandwidth, connectivity. With that have come a whole new breeds of internet users: commercial and re-creational. They see this as another whole resource to maximize, externalizing their costs, at the cost of every other user.

Many of the commercial users treat it with the same regard they do the natural world: less than respectful. Many of the recreational users are not aware of the history, customs and 'nettiquette' on which it is based. Because of all these factors, internet security has become much more prominent, comprising of machine and network security. Many are familiar with machine security, by limiting access to machines via password logins and the importance of picking passwords that cannot be dictionary attacked.

Network security is a more recent urgency, because packet sniffers are now much more accessible to the public in the form of inexpensive or free software packages that are able to turn common hardware into useful passive and now active analysis tools. It is for these reasons that network-layer encryption has become more and more important. IPSec is a suite of RFC proposed standards that define a protocol for packet-layer encryption.

Why use Linux? Linux is inexpensive, popular, and runs on a wide variety of hardware and processor families. Proprietary kernels are well documented, but not always documented correctly. Open-source kernels are not particularly well documented, but that is changing. The big advantage is that if something doesn't exist or doesn't work as (not) documented, you can find out how to use it, or send a patch to fix it.

What problems did I (we) have and how did we get around them? That is the subject of this presentation after we get some basic concepts out of the way.

The Author

Richard Guy Briggs got his taste of Un*x-like systems in 1990 while at Corel Systems Corporation, testing interoperation issues with SCO ODT 1.0 on 80386, Solaris on a Sparc IPX and the 16-bit version of Coherent on a 80286 and at the University of Ottawa on DEC Ultrix and IBM AIX systems. The _Jargon File_ was a significant influence.

He has been working with Linux since version 0.13 when he saw internet announcements about it on comp.ox.minix. He subsequently used it to train the artificial neural network temporal integrator for his 4th year undergraduate speech recognition project in September 1992.

Two solar vehicle competitions later (one in the USA and one in Australia), he started maintaining and porting device drivers for ISA-bus telecommunications cards (T1, E1, ISDN) under SCO, UnixWare and SolarisPC.

Richard has been maintaining and updating KLIPS, the FreeS/WAN kernel module for that Linux IPSec implementation for 18 months.

For more background, see: http://www.conscoop.ottawa.on.ca/rgb/cv.html or email him at: rgb@conscoop.ottawa.on.ca. More information about FreeS/WAN can be found at http://www.xs4all.nl/~freeswan/.

MERGEMEM

Friday, 10.9., 12:15-13:00

Philipp Reisner
Ulrich Neumerkel
Philipp Richter

1 Motivation

The gap between memory access time and hard-disk's seek time is becoming bigger every year. The same holds true for CPU speed and hard-disk speed. In this scenario, the traditional way of freeing pages in a situation of memory shortage, which is saving some pages to disk, is increasingly becoming problematic.

The alternative is to find an in-memory solution. One option is using compression algorithms.

But both, the traditional and the compression approaches, share a common disadvantage: If the process is accessing non-present pages, the operating system must restore them and thus eventually needs to page out other pages of main memory. Mergemem, in contrast, tries to increase the amount of shared pages, thus allowing read-only access to the treated pages without any overhead.

2 Memory sharing

Today's operating systems, including Linux, are using page sharing to save main memory. Code segments of executables and shared libraries are shared by all processes, thus mapped to various VM spaces but mapped only once to physical memory.

When a process calls fork(2), also the data segments (initialized data, uninitialized data, heap and stack) are shared between the process and the newly created child. If one of the two processes writes to the shared area, the operating system is trapped and creates a private copy for the writer (=copy on write).

Unfortunately a lot of processes running the same program do not have the chance to share their data segments, because they are not created by forking. Examples for this are in.telnetd processes created by inetd, or xterms created by a window manager.

Another problem is that programs tend to initialize their data structures in some way, thus the pages get copied for private use, but after that initialization phase the pages actually carry the same contents. A typical example of this problem is an interpreter building some representation of an included file.

Mergemem

Mergemem finds equal pages in running processes, frees the redundant pages, and gives all participating processes a read only reference to the remaining page. The running processes are not affected, and if there is a write operation to a shared page, the normal copy-on-write takes place.

3.1 mergemod, the kernel module

Mergemod is a character device which provides two important functions to the system.

- It provides the ability to look at the memory of other processes, without violating system security.
- It carries out a page merge, but rejects requests if the pages do not carry the same contents. We had to make shure that the pages are not changed inbetween the byte-by-byte comparison and merging.

Thus mergemod is responsible for correctness.

3.2 mmlib, the clue layer

We think that a stable API is as important as the systems itself, so we designed the mmlib interface with the following goals in mind. The interface should

- hide the interface to the kernel part (device & ioctl)
- hide the page size (Maybe we will see architectures without strictly fixed page size in future?)
- not predetermine search heuristic and checksum algorithm.

3.3 mergemem, the user-level-program

Mergemem searches for equal pages and triggers the merging of these. It turned out to be useful to have a single-shot mode and a daemon mode of mergemem.

In order to facilitate the search for pages with equal contents, mergemem calculates the checksums of all relevant pages and thus reduces the search to finding equal checksums. We noted that the crc32 checksum algorithm has a hit ratio of 100% in our tests, but we believe that there is a cheaper checksum algorithm that delivers a reasonable hit rate.

In the current implementation the search and checksum functions are plug-ins and thus very easy to change.

Mergemem is responsible for efficiency.

4 Impact on architectural level

From the view of the computer architecture, the effect of mergemem is the same as that of operating systems using copy-on-write strategies.

Today virtually every architecture can deal with pages mapped to more than one VM space in an efficient way.

When mergemem shares a page:

- physically indexed L1 cache Performance increases, since the system acts more locally.
- all kinds of virtually indexed L1 cache Performance remains unchanged. No consistency problems arise, because the mappings are read only.
- distributed caches in SMP systems Theoretically performance remains unchanged.

5 Status

The first implementation was based on Linux 2.0 and checksum calculation was carried out in kernel mode. It was in use at the Vienna University of Technology for a prolog class for about 4 months. The installation consists of about 20 X-terminals connected to a single server, and every student ran emacs and prolog. Mergemem was a great success in this environment, but unfortunately we were not allowed to collect performance data during the term. We are currently working on a new implementation as described in Section mergemem, and it is close to release (probably in August). It is based on Linux 2.2 and already supports SMP systems.

6 Sample Results

* Spread Sheet from Java JDK 1.1.3

1st Instance	12488KB
further instance	+7612KB
merging	-5160KB

* SICStus Prolog

1st Instance	2216KB
further instance	+964KB
merging	-880KB

7 Application scenarios

The amount of saved memory depends to a high degree on the running applications. At a typical single user Linux workstation with only two running xterms, besides the usual daemons, X, mingetty, etc., about 1 MB of memory is saved.

But since single user workstations require high administrative efforts, there is a tendency to use central application servers in combination with diskless terminals. In this environment, as was showed by our prolog class, mergemem is able to double the available memory.

8 References

http://www.ist.org/mergemem/,
http://www.complang.tuwien.ac.at/~ulrich/mergemem/

Netfilter: Packet Mangling in 2.4

Friday, 10.9., 14:30-15:15
Paul "Rusty" Russell

The 2.4 kernel will have significant enhancements in its networking code. In particular, the masquerading, port-forwarding, transparent-proxying and packet filtering systems have been formalized on top of a framework called netfilter. Netfilter developer, Paul "Rusty" Russell will describe in depth what this means for kernel developers and users.

WINE

Friday, 10.9., 14:30-15:15
Marcus Meissner

The free Windows Emulator WINE is one of the longest standing projects associated with Linux. While it is still in ALPHA development state, a lot of windows applications already do start and are partially useable.

Started in 1993 as an attempt to run Windows 3.1 binaries on Linux most of the work currently done is to implement the Win32 API on top of UNIX and X11, with the goal to run Windows Binaries (16 and 32 Bit) and to provide sourcecode compatibility for windows applications on UNIX platforms.

While WINE is no longer the way to get applications to Linux (there already is a huge set of native ones), it is not obsolete. WINE is needed for legacy applications (both 16 and 32 bit), applications that won't be ported natively to Linux, be it due to political reasons, lack of money or interest on the companies.

Speed of development has been increasing steadily over the last years, but it increased by nearly an order of magnitude starting the involvement of Corel which announced porting its Office Suite using WINELIB at the end of 1998. With approximately 20 fulltime developers working on WINE they contribute a rather large part of work on the additional libraries, like OLE and Common Controls. Their goal is to have all of their Office Suite Applications ported using WINELIB by the end of 1999.

A large part of the WINE developers (both volunteers and Corel employees) are currently focusing on getting the Win32 core functionality working, including multithreading, Win32 synchronization objects, and multiple win32 process support. Others are working on replacements for Windows DLLs, like OLE, Common Controls, DirectX, and Postscript Printer Support.

While it is difficult to give an estimate time of when WINE will be finished, it

can be said that by the end 1999 a large number of applications will run sufficiently well to be able to use them.

The talk will give a brief introduction and historic overview over the project, a technical introduction on how it is possible to run Win16 and Win32 binaries on a 32bit UNIX systems, followed by a report on the current status of the project (including problems), some demonstrations and should have room for questions and discussion.

NIS and NIS+ for Linux

The protocol, implementation and plans for the future

Friday, 10.9., 15:15-16:00
Thorsten Kukuk

A lot of people know about NIS and NIS+ and use one or the other for their daily work. For this reason, my talk will not focus on how to install and configure it. Instead I would like to talk about the underlying protocol, the advantages and the problems of both NIS and NIS+. This should help the user to solve problems faster. I will also tell something about the current status of the NIS/NIS+ implementations for Linux and my plans for the future. NIS is the older protocol of both. Nearly every Unix has support for it. There even exists a client for Windows NT.

Unfortunately, Sun has only documented the communication between the server (ypserv) and the client (C-library) for NIS. The protocol between client and ypbind is not openly documented, but with the header files the important parts of the ypbind protocol versions 1 and 2 could be implemented.

At this point I will tell something about the different client implementations for Linux (libc5, libc5+NYS, glibc) and the daemons (ypbind/ypserv). I will also talk about the different solutions for shadow over NIS and how ypserv can handle more than one domain. One disadvantage of NIS is, that we have only one central master server. For every single change the whole map has to be transferred to the slave, and an own table is needed for every searchable column.

NIS+ is the successor of NIS and partially solves its problems. On the client side we have a relatively good documenation, it was not a big problem to implement the NIS+ functions for glibc, on the contrary there is almost no documentation about the protocols between two NIS+ servers and between the administration tools and a server. We have only some names and parameters of the RPC function calls.

Since there where a lot of changes for NIS+ from Sun, I will refer to the current version in Solaris 7. I will explain the protocol between client and server and how the Linux client finds the server. Concering the security: NIS+ does not encrypt all the data it should, but with GSSRPC we have now a solution for the much too short keys of Secure RPC.

Future plans:

- nis_cachemgr for clients
- prefer NIS+ server in local subnets, nisprefadm
- Secure RPC with bigger keys (GSSRPC ?)
- NIS+ server for Linux
- implement missing features of NIS+ tools

The LinuxDirector Virtual-Server Project

All for one and one for all

Friday, 10.9., 15:15-16:00
Lars Marowsky-Brée

E-Commerce websites - the new challenge which we need to meet:

- Highly computing intensive tasks like Java servlets with heavy database interaction require more CPU power than a single system can provide.
- Processing power needs to scale easily to satisfy the increasing demands of success.
- Availability is a must.
- Security is an absolute must.
- It is preferrable if the solution is mostly transparent to the customer application.

In the past, solutions like the Cisco LocalDirector, F5 BIG/ip and others have been used to provide transparent load balancing between multiple servers. However, besides the price tag, these solutions are all proprietary and often do not fullfill all the demands at once.

I will present the problems and benefits encountered with Linux as a webscaling solution. I will try not to focus tightly on the theoretical aspects, but also point out the issues encountered in the field during actual implementation and deployment.

The "Linux VirtualServer" project by Wensong Zhang now provides a useable framework upon which sophisticated solutions can be build. Combined with the advanced routing fea-

tures of Linux, advanced packet filtering techniques and the power of the freely available applications, Linux rivals and even exceeds the most sophisticated solutions.

We are implementing such a solution for a customer. I will discuss the most convincing arguments in favor of the Linux solution, give a summary of the network and application design and then delve into the implementation process.

We will discuss the patches and tools used to build the redundant load balancing system itself and the webservers. In particular, the load balancer uses sophisticated network monitoring to intelligently distribute the load to the servers, uses the heartbeat package to implement the HA failover, automatically replicates the configuration to the hot-standby system and out-of-band communications.

I will also comment on the various features which we found most useful in implementing the whole cluster setup - content replication via CODA, completely packaged systems, easy replacement of failed systems using a KickStart mechanism, serial console remote control etc.

At the time of this abstract, the contract was just signed and you will be the first to know the actual story, and especially the issues we had to solve on the way. We will present first performance measurements and operational results.

The final conclusion will be whether we made it or not.