Perl Mongers im Ruhrgebiet

Ruhr.pm

Fuse mit Perl

Autor: Simon Wilper

EMail: simon AT ruhr.pm.org

Datum: 11. Februar 2009

http://ruhr.pm.org/

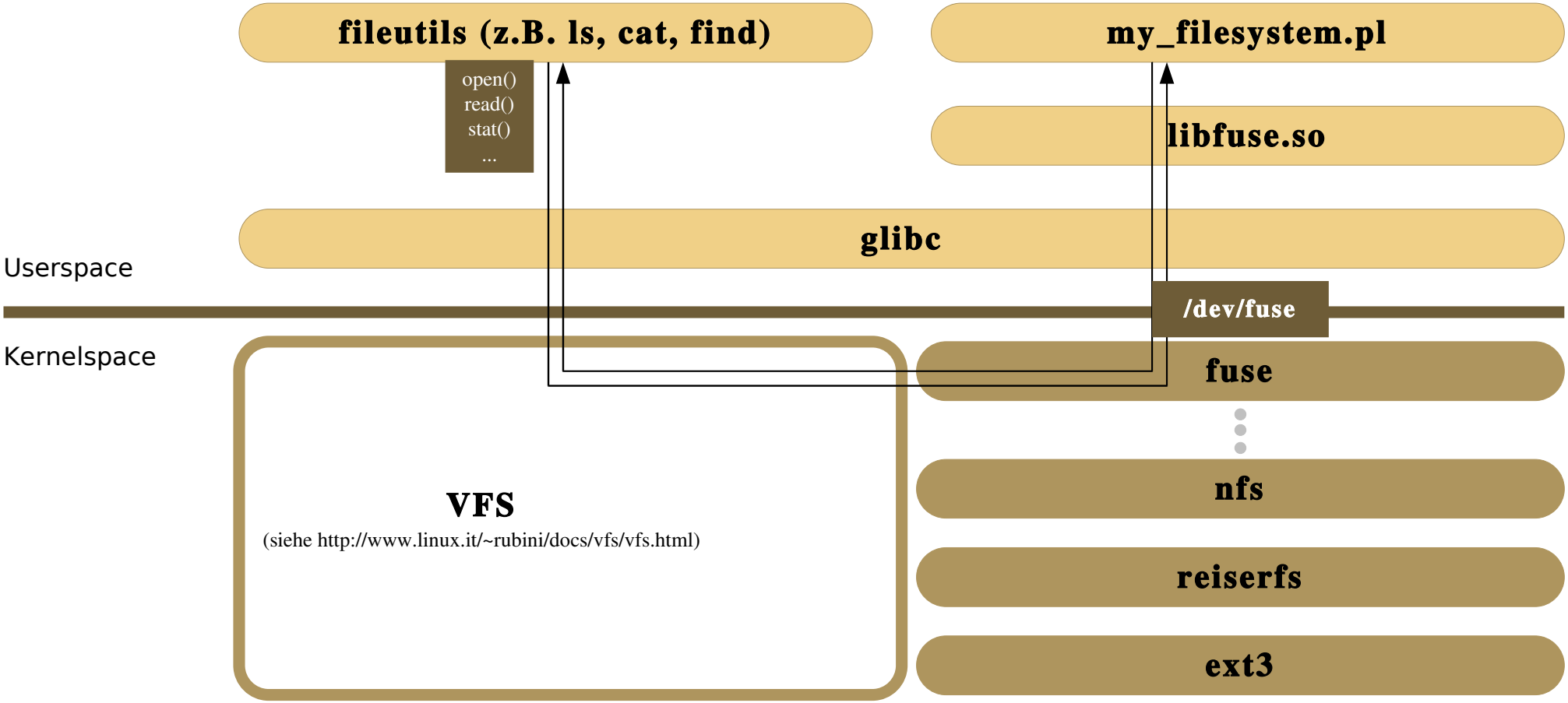
Übersicht

- Einführung
  - Was ist Fuse?
  - Wie funktioniert es?
  - Was gibt es bereits?
- Erste Schritte
  - Fuse::main
  - Was man mindestens brauch
- Beispiele
  - Erstmal was Generisches
  - ProcKillFS
  - SendmailFS

Was ist Fuse?

- Filesystem in Userspace
  - http://fuse.sourceforge.net/
- stellt beliebige Daten hierarchisch entsprechend dem Verständnis eines klassischen Dateisystems aus Verzeichnissen und Dateien dar
- benötigt Kernelmodul „fuse“
- User-/Kernelspace-Kommunikation via /dev/fuse

Wie funktioniert es?



Vor-/Nachteile

+ Schnelles Visualisieren der Daten -> Erspart das Entwickeln einer GUI
+ Datenmanipulation durch schon vorhandene Tools (fileutils, Dateimanager (mc, nautilus, thunar))
- Kernelanpassungen notwendig
- Kernelmodul muss eventuell neu kompiliert werden mit jedem Kernelupdate

Was gibt es bereits?

- Encfs: verschlüsselte Datein und Verzeichnisse
  - http://www.arg0.net/encfs
- SSHFS: SSH-Verbindung lokal mounten
  - http://fuse.sourceforge.net/sshfs.html
- FTPFS: FTP-Verzeichnis lokal mounten
  - http://ftpfs.sourceforge.net/
- GMailFS: Google-Mail als Dateisystem benutzen
  - http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html

Erste Schritte

- Fuse initialisieren
- Registrieren von Hooks, die das eigene Dateisystem implementieren
- Filesystem soll zunächst eine statische Liste ausgeben

Fuse::main

```
#!/usr/bin/perl

use POSIX;
use Fuse;

sub my_getdir {
        return -1*ENOENT;
}

sub my_getattr {
        return -1*ENOENT;
}

Fuse::main(
        mountpoint      => '/mnt',
        getdir          => \&my_getdir,
        getattr         => \&my_getattr,
        debug           => 1
);
```

POSIX: Konstanten (ENOENT, etc)

Fuse: Haupt-Perl-Modul

Fuse::main: Main-Methode, die als Parameter den Mountpunkt, alle Hooks nimmt, die den zu implementierenden syscalls entsprechen und eine Variable, das Debugging zu aktivieren

In diesem Beispiel werden die Hooks registriert:

getdir: Verzeichnis auslesen
getattr: Dateiattribute zurueckgeben

Dieses Beispiel macht noch nicht viel, gibt einfach ENOENT („No such file or directory“) zurück

Was man mindestens brauch: getdir ...

```
sub my_getdir {
        return (
                '.',
                'Bratwurst',
                'Mettwurst',
                'Teewurst',
                0
        );
}
```

getdir-Hook: Verzeichnislisting gibt eine Liste des Inhalts zurueck.

'.' als das aktuelle Verzeichnis

0 als das letzte Element, um die Liste zu terminieren

... und getattr

```
sub my_getattr {

        my $mode = 0040 << 9 | 0755;

        my $nlink = 1;

        my $uid = $<;
        my ($gid) = split / /, $(;

        my $size = 0;
        my $rdev = 0;

        my $atime = time;
        my $mtime = $atime;
        my $ctime = $atime;

        my $blksize = 1024;
        :
```

getdir-Hook: Attribute (Metadaten) zu den Dateien zurückliefern

$mode beinhaltet den Typ (0040=Verzeichnis, 0100=Datei (siehe man 2 stat)) um 9 bits nach links geshiftet und mit den Zugriffsrechten in oktaler Notation.

$nlink: Anzahl der hard links

$uid, $gid: User-ID und Gruppen-ID (ist die erste in $((alle Gruppen, die der Benutzer angehoert sind durch ein Leerzeichen getrennt))

$size: Dateigröße (wird hier erstmal auf 0 gesetzt)

$rdev: Device-ID (Für special files)

$(a|m|c)time: Access, Modification, Creation time zur Einfachheit auf time gesetzt

$blksize: Blocksize

getattr ctd.

```
        my $blocks = 1;
        my $dev = 0;
        my $ino = 0;

        return (
                $dev, $ino, $mode, $nlink,
                $uid, $gid, $rdev, $size,
                $atime, $mtime, $ctime,
                $blksize, $blocks
        );
}
```

$blocks: Anzahl Blöcke
$dev: Device auf dem die Datei liegt
$ino: Inode (besser auf 0 lassen)

Dieses Filesystem macht keine Unterscheidung zwischen Dateien und Verzeichnissen; es wird immer Verzeichnisse ausgeben, egal, in welchem man sich gerade befindet:

```
Terminal - sxw@tele:~/tmp
[sxw@tele] [~/tmp] > ls mnt -l
total 1,5K
drwxr-xr-x 1 sxw mco 0 11. Feb 03:43 Bratwurst
drwxr-xr-x 1 sxw mco 0 11. Feb 03:43 Mettwurst
drwxr-xr-x 1 sxw mco 0 11. Feb 03:43 Teewurst
[sxw@tele] [~/tmp] > ls mnt/Bratwurst/ -l
total 1,5K
drwxr-xr-x 1 sxw mco 0 11. Feb 03:44 Bratwurst
drwxr-xr-x 1 sxw mco 0 11. Feb 03:44 Mettwurst
drwxr-xr-x 1 sxw mco 0 11. Feb 03:44 Teewurst
[sxw@tele] [~/tmp] >
[sxw@tele] [~/tmp] >
```

Erstmal was Generisches

- Eine Perl-Datenstruktur (Hashes) auf einem Dateisystem abbilden

```
my $filesystem = {
 root => {
  content => {
   wurst => {
    type => 'dir',
    content => {
     bratwurst => {
      type => 'dir'
     },
     bockwurst => {
      type => 'dir',
      content => {
       gebraten => {
```

$filesystem: Eine Perl-Datenstruktur aus hashrefs wird als FS abgebildet.

root: Wurzelverzeichnis

Alle Elemente haben einen type-Key (file oder dir) und einen optionalen content-Key, der wiederum eine hashref auf weitere Verzeichnis enthaelt oder im Falle einer Datei den Dateiinhalt.

Angepasste getdir-Methode

```
sub my_getdir {
 my ( $filename ) = @_;
 print "getdir requesting [$filename]\n";

 my $current = $filesystem->{'root'}->{'content'};
 my @pathElements = split( '/', $filename );

 if ( @pathElements > 1 ) {
  foreach my $pathElement (
   @pathElements[1..$#pathElements] ) {

   return( -1*ENOENT ) if (
    !defined( $current->{$pathElement} ) );

   current = $current->{$pathElement}->{'content'};
  }
 }
 return( '.', keys( %{$current} ), 0 );
}
```

Die Pfad-Elemente werden gesplittet und ueber das Array iteriert.

Dabei wird der aktuelle Node im Hashref-Tree bei jedem Durchlauf auf $current gelegt. Am Ende der Path-Element-Iteration liegt der angeforderte Pfadinhalt in $current und es koennen die keys zurueckgegeben werden, die dann den Datei-/Verzeichnisnamen entsprechen.

Angepasste getattr-Methode

```
my $current = $filesystem->{'root'}->{'content'};
my @pathElements = split( '/', $filename );
my $currentType = 'file';

if ( @pathElements > 1 ) {
        foreach my $pathElement ( @pathElements[1..$#pathElements] ) {
                if ( !defined( $current->{$pathElement} ) ) {
                        return -1*ENOENT;
                }

                $currentType = $current->{$pathElement}->{'type'};
                $lastAttredFile = $current->{$pathElement}
                        if ( $currentType eq 'file' );

                $current = $current->{$pathElement}->{'content'}
                        if ( $currentType eq 'dir' );
        }
}

if ( $filename eq '/' || $currentType eq 'dir' ) {
        $type = 0040;
        $bits = 0755;
        print "DIR\n"
} else {
        print "FILE\n";
}
```

Wieder wird über die Pfadelemente iteriert.

Der aktuelle Typ des abgefragten Objekts wird in $currentType gespeichert, bevor $current auf den Inhalt des nächsten Kindelemts gesetzt wird.

$lastAttredFile ist eine globale Variable, in der die zuletzt angeforderte Datei liegt, damit man den Inhalt spaeter in read() einfacher ausgeben kann.

Dateien lesen: read()-Methode

```
if ( $currentType eq 'file' ) {
 $size = length( $lastAttredFile->{'content'} );
}
```

Wenn die Größe 0 ist, wird nichts uebertragen von read.

Also in der gettattr-Methode die Stringlänge als Größe zurückgeben.

```
sub my_read {
        my ( $filename ) = @_;
        return $lastAttredFile->{'content'};
}
```

read-Methode gibt den Dateiinhalt zurueck

```
Fuse::main(
        mountpoint      => '/mnt',
        getdir          => \&my_getdir,
        getattr         => \&my_getattr,
        read            => \&my_read,
        debug => 0
);
```

In Fuse::main den read-Hook belegen

Ruhr.pm

Das Ergebnis

Terminal - sxw@tele:~/tmp/mnt

```
[sxw@tele] [~/tmp/mnt] > ls
obst  wurst  info
[sxw@tele] [~/tmp/mnt] > tree
.
|-- info
|-- obst
|   `-- apfel
|       `-- wurm
`-- wurst
    |-- bockwurst
    |   |-- gebraten
    |   `-- geschmort
    |-- bratwurst
    `-- teewurst
        `-- liesmich

7 directories, 4 files
[sxw@tele] [~/tmp/mnt] > cat info
Wurst- und Obst-Filesystem V0.1
[sxw@tele] [~/tmp/mnt] > grep -Ri wurm *
obst/apfel/wurm:Hier ist der Wurm drin.
[sxw@tele] [~/tmp/mnt] > find -name 'lies*'
./wurst/teewurst/liesmich
[sxw@tele] [~/tmp/mnt] >
```

ProcKillFS

- Prozesse werden als Dateien aufgelistet und können gekillt werden, indem man die repräsentierende Datei löscht
- Auflistung nach UID
- Basis aus vorigem Beispiel:

```
my $filesystem = {
 root => {
  content => {
   0 => {
    type => 'dir',
    content => {
     1_init => {
      type => 'file'
     }
```

$filesystem: Im ersten Level sind die UIDs als Verzeichnisse angesiedelt.

Die enthalten wiederum eine Liste von Dateien, die den Prozessen der jeweiligen UID entsprechen in der Form:

$ProzessID_$ProzessName

Das ProcKillFS „befüllen“

```
sub updateProcs {
        $filesystem = { root => { content => {} } };
        my $procdir = IO::Dir->new( "/proc" );
        if ( !defined( $procdir ) ) {
                die ( "Unable to open proc dir" );
        }
        while ( defined( $_ = $procdir->read ) ) {
                next if ( /^\D/ );
                my $PID = $_; my $uid = 0; my $name = 'na';

                open ( my $fhstatus, "/proc/$PID/status" );
                while ( <$fhstatus> ) {
                        chomp;
                        if ( m/^Uid:\s+(\d+)/ ) { $uid = $1; }
                        if ( m/^Name:\s+(.+)$/ ) { $name = $1; }
                }
                close( $fhstatus );

                $name =~ s/[\:\-\@\/.\,*\]\[\s\)\(]/_/g;

                $filesystem->{root}->{content}->{$uid} = { type => 'dir', content => {} } if
                        ( !defined( $filesystem->{root}->{content}->{$uid} ) );

                $filesystem->{root}->{content}->{$uid}->{content}->{$PID.'_'.$name} = { type => 'file' };
        }
}
```

IO::Dir: Es wird über alle Verzeichnisse iteriert unterhalb von /proc, die einer PID entsprechen.

Die UID und der Prozessname werden aus der Datei „/proc/$PID/status“ ausgelesen

In unserem root-filesystem wird ein Verzeichnis entsprechend der UID angelegt und mit content=>{} ein leeres Verzeichnis initialisiert.

Anschließend werden die Prozesse in der Form $PID.'_'.$name als Dateien dem Verzeichnis hinzugefügt

my_unlink: Kein Warmduscher-Kill

```
sub my_unlink {
        my ( $filename ) = @_;
        my( $pid ) = ($filename =~ m/\/(\d+)_.+$/ );

        print "Killing: [$pid]\n";
        kill( 9, $pid );

        return 0;
}

updateProcs;

Fuse::main(
        mountpoint      => '/home/sxw/tmp/mnt',
        getdir          => \&my_getdir,
        getattr         => \&my_getattr,
        unlink          => \&my_unlink,
        debug => 0
);
```

my_unlink: Prozess-ID aus Dateinamen extrahieren und kill ausführen

updateProcs: Filesystem mit Prozess-IDs initialisieren

Fuse::main: unlink-Hook registrieren

Ruhr.pm

Ergebnis

Terminal - sxw@tele:~/tmp/mnt

```
[sxw@tele] [~/tmp/mnt] > ls
0  73  800  81  82  88
[sxw@tele] [~/tmp/mnt] > ls 800
15438_gvim              23953_firefox          5021_xfce4_fsguard_p
15472_Terminal          23956_run_mozilla_sh   5022_xfce4_fsguard_p
15474_gnome_pty_helpe   23962_firefox_bin      5023_xfce4_diskperf_
15475_bash              23968_gconfd_2         5024_xfce4_netload_p
15504_bash              23989_ls               5025_xfce4_systemloa
17960_soffice           4974_bash              5026_xfce4_mixer_plu
17970_soffice_bin       5011_ssh_agent         5027_xfce4_timer
20500_bash              5013_fetchmail         5028_xfce4_screensho
20966_bash              5015_xfce_mcs_manage   5029_xfce4_xkb_plugi
20986_rtorrent          5016_xfce4_panel       5037_dbus_launch
23758_prockill_fs_pl    5017_openbox           5038_dbus_daemon
23800_mousepad          5019_xfce4_menu_plug   8732_notification_da
23930_bash              5020_xfce4_battery_p
[sxw@tele] [~/tmp/mnt] > find -name '*firefox*'
./800/23953_firefox
./800/23962_firefox_bin
[sxw@tele] [~/tmp/mnt] > find -name '*firefox*' -delete
[sxw@tele] [~/tmp/mnt] >
```

Ruhr.pm

SendmailFS

- Email-Adressen aus einer beliebigen Datenquelle werden als Verzeichnisse dargestellt
- Neue Mails werden verschickt, indem in dem entsprechenden Mailadressen-Verzeichnis eine Datei angelegt wird, die die Nachricht enthaelt.
- Der Dateiname ist das Subject

```
my $filesystem = {
        root => {
                content => {
                        'undef@example.org' => { type => 'dir' },
                }
        }
};
```

Neue Hooks

- mknod: Anlegen von Dateien -- Mail wird als type=>'file' im FS-Tree angelegt
  - Parameter: Dateiname, Mode, Device-ID
- write: Schreiben -- Mail wird in einem Puffer zwischengespeichert
  - Parameter: Dateiname, Puffer, Offset
- flush: Schließen -- Mail wird via Net::SMTP gesendet
  - Parameter: Dateiname

mknod

```
sub my_mknod {
        my ( $filename, $mode, $dev ) = @_;
        print "MkNode $filename\n";

        my @pathElements = split( '/', $filename );
        $filesystem->{root}->{content}->
                {$pathElements[1]}->{content} = { $pathElements[2] =>
                                { type => 'file' }
        };

        $currentBody = '';
        $currentMailAddress = $pathElements[1];
        $currentSubject = $pathElements[2];

        return 0;
}
```

my_mknod: currentBody, ~MailAddress, ~Subject werden initialisiert.

Mail-Adresse aus erstem Pfadelement

Subject aus zweitem Pfadelement (Dateiname, der Datei, die gerade erstellt wird)

Body leer initialisieren

write

```
sub my_write {
        my ( $filename, $buffer, $offset ) = @_;

        print( "Writing [$filename] Offset: $offset\n" );
        print "Buffer: [$buffer]\n";

        $currentBody .= $buffer;

        return length( $buffer );
}
```

my_write: currentBody wird mit $buffer befuellt.

Rueckgabe ist hier die Anzahl der geschriebenen Bytes

flush

```
sub my_flush {
        my ( $filename ) = @_;
        print "Flushing: $filename\n";

        if ( $currentBody eq '' ) {
                return 0;
        }

        my $smtp = Net::SMTP->new( 'localhost' );
        $smtp->mail( 'juergen.vogel@polizeiruf.110' );
        $smtp->to( $currentMailAddress );
        $smtp->data();
        $smtp->datasend( "Subject: $currentSubject\n\n$currentBody" );
        $smtp->dataend();
        $smtp->quit();

        return 0;
}
```

my_flush: Wenn der Body nicht leer ist:

SMTP-Objekt instanzieren. Es wird angenommen, das auf dem lokalen System ein MTA laeuft.

Nachricht, die in $currentBody steht verschicken

Ruhr.pm

Kaffeepause

```
Terminal - sxw@tele:~/tmp/mnt
[sxw@tele] [~/tmp/mnt] > ls
christoph.schlingensief@googlemail.com  simon@ruhr.pm.org  simonaw@openoffice.org
[sxw@tele] [~/tmp/mnt] > for i in *; do echo 'Kaffee ist fertig!' > $i/Kaffeepause; done
[sxw@tele] [~/tmp/mnt] >
```

```
Terminal - sxw@tele:~
i:Exit  -:PrevPg  <Space>:NextPg v:View Attachm.  d:Del  r:Reply  j:Next ?:Help
    1 [N  ] 11.02.2009 15:13:  simon@ruhr.pm.org     - Kaffeepause
    2 [N  ] 11.02.2009 15:13:  simon@ruhr.pm.org     - Kaffeepause
-*-Mutt: /var/mail/sxw [Msgs:2 New:1 3,OK]---(reverse-date/date)--------(all)---
From simon@ruhr.pm.org  Wed Feb 11 15:13:50 2009
Subject: Kaffeepause
Date: Wed, 11 Feb 2009 15:13:16 +0100 (CET)
From: simon@ruhr.pm.org
To: undisclosed-recipients: ;

Kaffee ist fertig!
~
~
~
~
~
~
~
~
~
~
~
-N  - 2/2: simon@ruhr.pm.org      Kaffeepause                          -- (all)
```

Ruhr.pm

Ende! :)