Institute for System Programming of Russian Academy of Sciences

ISP RAS

founded in 1994

• Basic research in computer science
• Applied research and development in the interests of the industry
• Education activities
ISPRAS Partners

- Telelogic
- Gelato - GCC
- The Linux Foundation
- Samsung
- KLOCwork
Linux Verification Center

founded in 2005

- OLVER Program
- Linux Standard Base Infrastructure Program
- Linux Driver Verification Program
OLVER (Open Linux VERification)

- Analysis and formalization of LSB Core 3.1 requirements on Linux system interfaces
- Development of a specification-based open source Test Suite for LSB conformance and functional testing of Linux
Target System
Linux Standard Base (LSB)

- Developed by Linux Foundation (linuxfoundation.org) since 1998
- The first version 1.0 – June 2001
- Current version 4.0 – January 2009
- Platinum Members:
  - HP, IBM, Intel, Novell, NEC, Fujitsu
- Gold Members:
  - CA, Mandriva, Miracle, MontaVista, Red Hat, Sun Microsystems, Turbolinux, etc.
Requirements Elicitation

Mark up assertions while reading standard

\textit{mvcur.01} The \texttt{mvcur()} function outputs one or more commands to the terminal that move the terminal's cursor to \texttt{(newrow, newcol)}, an absolute position on the terminal screen. \textit{mvcur.02} The \texttt{(oldrow, oldcol)} arguments specify the former cursor position. \textit{mvcur.03.01} Specifying the former position is necessary on terminals that do not provide coordinate-based movement commands. \textit{mvcur.03.02} On terminals that provide these commands, Curses may select a more efficient way to move the cursor based on the former position. \textit{mvcur.04} If \texttt{(newrow, newcol)} is not a valid address for the terminal in use, \texttt{mvcur()} fails. \textit{mvcur.05} If \texttt{(oldrow, oldcol)} is the same as \texttt{(newrow, newcol)}, then \texttt{mvcur()} succeeds without taking any action. \textit{mvcur.06} If \texttt{mvcur()} outputs a cursor movement command, it updates its information concerning the location of the cursor on the terminal.
OLVER Process

1. Analysis of the Standard and Extraction of Requirements
2. Formalization of Testable Requirements
3. Development of Test Scenarios
4. Automatic Generation of Tests by CTesK Tools

Legend:
- Process
- Intermediate Result
- Result for Users
Universal test assertions are produced ready to paste into the source code of the test. Each assertion has correct ID and comment.

Pre/post conditions and errors are grouped separately.
Specification Example

```c
specification
CString* basename_spec(CallContext context, CString* path)
{
    pre
    {
        return true;
    }

    coverage C
    {
        if (path == NULL)
        {
            return (NullPtr, "Null pointer received");
        }
        else if (equals(path, create_CString(""))
        {
            return (Empty, "Empty path received");
        }
        else if (_)
        {
            ...
        }
    }

    post
    {
        /* If path is a null pointer or points to an empty string,
         * basename() shall return a pointer to the string ".".
        */
        if (@path==NULL)
        {
            REQ( "basename.04",
                "If path is null, basename() shall return ".", 
                equals(basename_spec, create_CString("."))
            );
            return true;
        }
        ...

        return true;
    }
}
```
Test Oracles

Automatic checking of output correctness

Test stimuli ➔ System Under Test ➔ Test oracle

Specifications ➔ System Under Test ➔ Test oracle
OLER Process
Test Scenario Model

Test Engine

Test Scenario
Olver Process

1. Analysis of the Standard and Extraction of Requirements
2. Formalization of Testable Requirements
3. Development of Test Scenarios
4. Automatic Generation of Tests by CTesK Tools

Legend:
- Process
- Intermediate Result
- Result for Users
Test Reports (1)

Expandable tree of all the assertions and overall statistics: assertion coverage for particular test run (covered/total)

- incurses.char.add (18 / 25)
- incurses.input.string (17 / 40)
- incurses.window.scroll (16 / 19)
- incurses.mode (15 / 22)
- incurses.terminal.control (13 / 77)
- incurses.input.timeout (11 / 11)
- incurses.move (10 / 17)
  - mvcur (4 / 9)
    - mvcur.01
      The mvcur() function outputs one or more commands to the terminal that move the terminal's cursor to (newrow, newcol), an absolute position on the terminal screen.
    - mvcur.02
      The (oldrow, oldcol) arguments specify the former cursor position.
    - mvcur.03.01
      Specifying the former position is necessary on terminals that do not provide coordinate-based movement commands.
    - mvcur.03.02
      On terminals that provide these commands, Curses may select a more efficient way to move the cursor based on the former position.
    - mvcur.04
      If (newrow, newcol) is not a valid address for the terminal in use, mvcur() fails.
    - mvcur.05
      If (oldrow, oldcol) is the same as (newrow, newcol), then mvcur() succeeds without taking any action.
    - mvcur.06
      If mvcur() outputs a cursor movement command, it updates its information concerning the location of the cursor on the terminal.
    - mvcur.07.01
      Upon successful completion, mvcur() returns OK.
Test Reports (2)

```c
#include <curses.h>

int mvcur(int oldrow, int oldcol, int newrow, int newcol);
```

**DESCRIPTION**

- **mvcur** function outputs one or more commands to the terminal that move the terminal's cursor to (newrow, newcol), an absolute position on the terminal screen.
- **arguments** specify the former cursor position.
- Specifying the former position is necessary on terminals that do not provide coordinate-based movement commands.
- On terminals that provide these commands, Curses may select a more efficient way to move the cursor based on the former position.
- If (newrow, newcol) is not a valid address for the terminal in use, **mvcur** fails.
- If (oldrow, oldcol) is the same as (newrow, newcol), then **mvcur** succeeds without taking any action.
- **mvcur** outputs a cursor movement command, it updates its information concerning the location of the cursor on the terminal.

**RETURN VALUE**

- Upon successful completion, **mvcur** returns OK. Otherwise, it returns ERR.

**ERRORS**

- No errors are defined.

**APPLICATION USAGE**

Green color marks requirements that were checked by the test run.
olver Results

- Open source test suite included in tto official LSB certification program
- LSB Infrastructure Program
LSB Infrastructure Program

http://ispras.linuxfoundation.org

- Test Development
- LSB Infrastructure Refactoring and Development
- Analytical Tasks
LSB Infrastructure Program

- LSB Navigator
- LSB Distribution Checker
- Linux Application Checker
- LSB Eclipse Plugin
- ABI Compliance Checker

New Test Development Technologies
- T2C
  - Tests for gtk stack, libstdc++, etc. (2000+ interfaces)
- Azov
  - Tests for Qt3, Qt4, libxml (21000+ interfaces)
Linux Driver Verification Program

- Target System Overview
- Program Goals and Architecture
- BLAST
- Current State and Future Plans
Linux Kernel Statistics (1)

• More than 1000 active developers

Linux Kernel Statistics (2)

• 3.83 patches per hour applied (avg for 4 years)

Linux Kernel Statistics (3)

- 6422 added/3285 removed/1687 modified every day

Linux Kernel
Device Drivers

• Up to 70% of kernel source code
• 85% kernel crashes comes from drivers


Linux Kernel
stable_api_nonsense.txt

This is being written to try to explain why Linux does not have a binary kernel interface, nor does it have a stable kernel interface. Please realize that this article describes the *in kernel* interfaces, not the kernel to userspace interfaces. The kernel to userspace interface is the one that application programs use, the syscall interface. That interface is very stable over time, and will not break.

**Executive Summary**

You think you want a stable kernel interface, but you really do not, and you don't even know it. What you want is a stable running driver, and you get that only if your driver is in the main kernel tree. You also get lots of other good benefits if your driver is in the main kernel tree, all of which has made Linux into such a strong, stable, and mature operating system which is the reason you are using it in the first place.

Greg Kroah-Hartman
Linux Kernel
Opportunities

- Static detection of kernel core API misuse and general programming errors in drivers
  - Complicated data structures are pretty limited
  - Floating point arithmetics are pretty limited
  - Recursive procedures are pretty rare
  - Size of drivers is limited

- “One button” solution required
Existing approaches

- Peter T. Breuer (University Politécnica de Madrid, Spain)
- H. Post, W. Kuchlin (Eberhard Karls University Tübingen, Germany)
- Thomas Witkowski (Dresden University), Nicolas Blanc, Daniel Kroening, Georg Weissenbacher (ETH Zurich) – DDVerify
- sparse – Semantic parser
Issues

- Just a case study for the own method/tools
- Issues with tools result analysis
  - No expertise in Linux kernel
  - Tools do not support gcc-specific extensions
  - No rewards for bug finders
- Kernel core over specification
Linux Driver Verification Program

- A platform for joint efforts
  - Collection of domain-specific rules to be verified
  - Framework for verification tools
  - Potentially, benchmark for verification tools
  - Expertise in Linux kernel for verification experts

- A tool for independent driver developers
  - They are outside of the kernel process
  - May be not familiar with all kernel core conventions and changes
LDV Tasks

- creating **verification framework** that supports multiple verification engines and applies the most appropriate one under the circumstances given
- building **repository** of typical problems in source code of Linux device drivers
- developing **domain specific verification engines** for Linux device drivers source code analysis
Verification Framework

Driver sources
Kernel sources
Formalized Verification Rule

Verification Engine 1
Verification Engine 2
Verification Engine 3
Verification Framework

- Driver sources
- Kernel sources
- Formalized Verification Rule

Verification Engine 2

[Diagram showing the integration of driver, kernel sources, and formalized verification rule into a verification engine 2]
Verification Rule

A verification rule is

- a requirement of absence of a specific potential problem
- a correctness requirement common for drivers of specific type
Potential Problems

- Generic programming bugs
  - Buffer overflow, NULL dereference, etc.
  - Deadlocks, race conditions
- Domain-specific problems
  - Kernel core API misuse
  - Performance recommendations, etc.
Repository of Verification Rules

- 71 rules identified
Repository of Verification Rules

- 71 rules identified
- 54 rules can be formalized as a reachability problem
Reachability

- Limitation on sequence of actions

```c
int ldv_lock = 0;
unsigned ldv_spin_lock_irqsave(spinlock_t *lock) {
    ldv_assert(ldv_lock==0);
    ldv_lock=1;
    return ldv_undef_ulong();
}

void ldv_spin_unlock_irqrestore(spinlock_t *lock,
                                unsigned flags) {
    ldv_assert(ldv_lock!=0);
    ldv_lock=0;
}
```
Reachability

- Limitation on sequence of actions
  - lock before unlock
  - create before use
  - do not use after free

- Usual assert
  - BUG_ON(), WARN_ON()
Reachability

- Limitation on sequence of actions
  - lock before unlock
  - create before use
  - do not use after free

- Usual assert
  - BUG_ON(), WARN_ON()

- Resource deallocation at some point
  - unlock all mutexes before return to kernel
Reachability (2)

```c
int main(int argc, char* argv[])
{
    ...
    assert();
    ...
}
```
static struct pci_driver DAC960_pci_driver = {
    .name = "DAC960",
    .id_table = DAC960_id_table,
    .probes = DAC960_Probe,
    .remove = DAC960_Remove,
};

static int DAC960_init_module(void)
{
    int ret;

    ret = pci_register_driver(&DAC960_pci_driver);
#ifdef DAC960_GAM_MINOR
    if (!ret)
        DAC960_gam_init();
#endif
    return ret;
}
...

module_init(DAC960_init_module);
module_exit(DAC960_cleanup_module);
Main Generator

```c
int main(int argc, char* argv[]) {
    init_module();
    for(;;) {
        switch(*) {
            case 0: driver_probe(*,*,*);
            case 1: driver_open(*,*)
            ...
        }
    }
    exit_module();
}
```
Main Generator - Issues

- Sequence limitations
  - open() after probe() but before remove()

- Implicit limitations
  - read() can be only if open() return 0

- All above is specific for each kind of devices
How to Formalize a Rule?

- assert inline in the target code?
- replace interesting functions in headers:

```c
#define spin_lock_irqsave   ldv_spin_lock_irqsave
#define spin_unlock_irqsave ldv_spin_unlock_irqsave
```

and add model implementation
Repository of Verification Rules

- 71 rules identified
- 54 rules can be formalized as a reachability problem
- ~30 rules can be formalized using preprocessor
What is a Problem with Others?

- Access to variables
- Access to fields of structures
- Dereferencing
- Casting
- GCC attributes
Aspect Oriented Approach

- Formalized verification rule is an aspect
  - Cutpoint – where to make a change
  - Aspect – how to change source code
- There is no good tool for C
  - AspectC, C4, AspectC++
- We develop a new one based on llvm-gcc
Repository of Verification Rules

- 71 rules identified
- 54 rules can be formalized as a reachability problem
- ~30 rules can be formalized using preprocessor
- 14 actually formalized
BLAST Authors

- **BLAST 1.0**
  - Thomas Henzinger
  - Rupak Majumdar
  - Ranjit Jhala
  - Gregoire Sutre

- **BLAST 2.0**
  - Dirk Beyer
BLAST History

- Nov 2003 → blast 1.0
- Oct 2005 → blast 2.0
- Aug 2007 → blast 2.4
  - Interpolation procedure separated from the model-checking engine
  - Theorem-prover interface based on the SMT-LIB expression language
- Jul 2008 → blast 2.5
  - We have integrated the new interpolation procedure CSIsat
BLAST History (2)

- 05.2008 → B2 pre-alpha
  - The re-implementation emphasizes cleaner design and exendible APIs over speed, making it a better test-bed for software verification
  - fork of blast 2.0?
■ Grégoire Sutre
  □ PhD 2000. Abstraction et accélération de systèmes infinis
  □ till 2001 → ENS de Cachan, France
  □ 2001 → Laboratoire Bordelais de Recherche en Informatique

■ Thomas A. Henzinger
  □ PhD 1991. The Temporal Specification and Verification of RealTime Systems
  □ 1996-2005 → Berkeley
  □ 2005 → EPFL IC Faculty, in Lausanne, Switzerland
  □ 2009 Sep. → Institute of Science and Technology (IST) Austria (President)

■ Rupak Majumdar
  □ PhD 2003/ Symbolic Algorithms for Verification and Control
  □ 2004 → University of California, Los Angeles

■ Ranjit Jhala
  □ PhD 2004. Program Verification by Lazy Abstraction
  □ 2005 → University of California, San Diego
Verification Framework

- Driver sources
- Kernel sources
- Formalized Verification Rule

BLAST
BLAST Results

on 14 rules (CSIsat, yices)

- false positives because of model over simplification
- parser issues
- internal exceptions
- timeouts
- real bugs
## Issues Detected

http://linuxtesting.org/results/ldv

- 13 issues reported, 11 approved and fixed

### Problems in Linux Kernel

This section contains information about problems in Linux kernel found within Linux Driver Verification program. Click on a problem number for detailed description. Click on a column header to change the sorting order.

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0009</td>
<td>Crash</td>
<td>isicom.c sleeping function called from invalid context</td>
<td>2009-10-08</td>
<td><a href="http://lkml.org/lkml/2009/10/7/246">http://lkml.org/lkml/2009/10/7/246</a></td>
<td>Recognized as an error.</td>
</tr>
</tbody>
</table>
Example of a report (L0009)

Date       Wed, 7 Oct 2009 14:31:32 +0100
From       Alan Cox <>
Subject    Re: [BUG] isicom.c sleeping function called from invalid context

On Wed, 7 Oct 2009 17:15:14 +0000
Alexander Strakh <strakh@koras.ru> wrote:
> KERNEL_VERSION: 2.6.31
> DESCRIBE:
> Driver drivers/char/isicom.c might sleep in atomic context, because it calls
tty_port_xmit_buf under spin_lock.
>
> ./drivers/char/isicom.c:
> 1307 static void isicom_hangup(struct tty_struct *tty)
> 1308 {
>    ...
> 1315         spin_lock_irqsave(&port->card->card_lock, flags);
> 1316     isicom_shutdown_port(port);
>    ...
> }
>
> Path to might_sleep macro from isicom_hangup:
> 1. isicom_hangup calls spin_lock_irqsave (drivers/char/isicom.c:1315) and then
calls isicom_shutdown_port.
> 2. isicom_shutdown_port calls tty_port_free_xmit_buf at
drivers/char/isicom.c:906
> 3. tty_port_free_xmit_buf calls mutex_lock at drivers/char/tty_port:48
>
> Found by Linux Driver Verification Project

Diagnosis is correct. I'll take a quick look at that one.
Example of a report (L0014)

Date: Mon, 12 Oct 2009 11:25:13 +0200 (CEST)
From: Jiri Kosina <>
Subject: Re: [BUG] hidraw.c: double mutex_lock

On Mon, 12 Oct 2009, iceberg wrote:

> KERNEL_VERSION: 2.6.31
> DESCRIBE:
>   In driver ./drivers/hid/hidraw.c in function hidraw_read may be
>   double mutex_lock:
>   
>   Path:
>   1. line 50: begin first iteration of “while(ret==0)”
>   2. line 52: first call to mutex_lock
>   3. inner loop “while (list->head == list->tail)” does not change state
>      of mutex, because mutex_lock immediately follows mutex_unlock
>   4. if we go to the second iteration of “while(ret == 0)” in
>      line 50 then there are second call to mutex_lock in line 52 (mutex
>      acquired twice).
>   
> Second iteration of loop “while(ret==0)” is possible if local variable
> ret is not changed at line 94: ret+=len - i.e. len==0;
> Variable len may be zero if hidraw_read is called with count==0 or
> list->buffer[list->tail].len == 0.

Good catch. I will fix that up by moving the mutex_lock() so that it’s
locked before the loop is entered.

Thanks,

--
Jiri Kosina
SUSE Labs, Novell Inc.
BLAST Issues

- Parser issues
- Internal exceptions
- Timeouts
- Hard to maintain code
- LLVM output is not good input for BLAST
Current state

- Verification repository
- Prototype of the verification framework
  - Main generator
  - Aspectator
- BLAST minor fixes and improvements
- Test run analysis and bug reporting
Future Plans

- BLAST improvements
- Public verification rules repository
- Alternative verification engines
  - CBMC
  - Non reachability
  - Multithreaded
- Partners cooperation
BLAST Improvements

“The re-implementation emphasizes cleaner design and exendible APIs over speed, making it a better test-bed for software verification” (B2' authors)

- CPA, CPA+
- Specialized extensions
  - Sets, ...
- Optimization
  - Profiling
  - Path invariants, Large-Blocks, Shape analysis
  - LLVM-related issues
Verification Rule - Example

Informal Description

An element of a linked list should be inserted in the list only once

If the list before operation looks like:

→a↔b↔c↔d↔e←

and we add the element \( c \) after the element \( a \), we will have the list:

→a↔c↔b→c { ↔b } ←d←

where \{ \} means the new link of the element \( c \).
Verification Rule - Example

Informal Description (2)

An element of a linked list should be inserted in the list only once

```
linux-2.6.25/drivers/firmware/dmi_scan.c:
void dmi_save_oem_strings_devices(dmi_header *dm) {

...  
for (i = 1; i <= count; i++) {
    char *devname = dmi_string(dm, i);
    if (!strcmp(devname, dmi_empty_string)) {
        list_add(&empty_oem_str_dev.list, &dmi_devices);
    }
...}
```
Verification Rule - Example

Failure Model

- State variable:
  ```c
  list_head *elem = NULL;
  ```

- Insert before call to `list_add( new, head)`
  ```c
  assert(new!=elem);
  if(*) elem = new; // * - nondeterministic choice
  ```

- Insert before call to `list_del( new, head)`
  ```c
  if(entry==elem) elem=NULL;
  ```
Thank you!

Alexey Khoroshilov
khoroshilov@linuxtesting.org
Web: http://www.linuxtesting.org
Vadim Mutilin
mutilin@ispras.ru