SIL4Linux: An attempt to explore Linux satisfying SIL4 in some restrictive conditions

Lijuan Wang, Chuande Zhang, Zhangjin Wu, Nicolas Mc Guire and Qingguo Zhou
Distributed and Embedded System Lab
School of Information Science and Engineering Lanzhou University
Tianshui South Road 222,Lanzhou,P.R.C
zhangchd.lzu.edu@gmail.com

Abstract

Linux is an existing widely-used operating system in lots of fields, including desktop applications, server solutions, embedded systems and even some real time controlling environment with the rt-preempt extensions. And it works well without any big problem currently, but for it is a complex and large system, some potential uncertain factors may influence its stability, so there is no guarantee to use it in some safety-critical environment.
In this paper, we will try to explore the possibility of Linux satisfying SIL 4 in some restrictive conditions. To achieve such a goal, a sil4linux system have been designed and implemented via integrating some kernel tracing/profiling tools, two formal analyzing methods, and with the support of a DBMS.

1 Introduction

In the modern world mankind depends on quite a number of safety-critical systems. Sometimes this dependability is obvious as on the subject of aircraft construction and sometimes it is hidden in tiny embedded systems like ones that trigger a fire alarm. Reference[1] states requirements to safe operating systems assessing Linux for safety related systems. Reference[2] point out some main issue of using COTS/OSS software in the context of 61508 compliant safety related systems, and also it is the initial idea of the whole SIL4linux project.
The SIL4linux project is attempt to find some available methods to explore the possibility that Linux is suitable for use in many safety related applications with SIL 4 integrity requirements.
In this article, we outline an architecture of the primary step of the sil4linux system, and its implementation, usage,related tools and try to outline the possible method to progress on the whole project. It should be noted that this paper just gives the first step of the SIL4Linux project to explore the possibility of using Linux or modified versions of Linux for SIL 4 applications.

2 Background

Before showing what and how we have done on the SIL4Linux project, we should interpret some basic concepts. The following sections will introduce SIL, SIL4, FMEA and FTA.

2.1 SIL

A SIL[3] is a measure of safety system performance, or probability of failure on demand (PFD) for a SIF or SIS. There are four discrete integrity levels associated with SIL. Both ISA and IEC have agreed that there are three categories: SIL 1, 2 and 3. IEC also includes an additional level SIL 4 that ISA does not. The higher the SIL level, the lower the probability of failure on demand for the safety system and the better the system performance. SIL 4[3] is the highest level of risk reduction that can be obtained through a Safety Instrumented System.

2.2 FMEA

FMEA[4] is a procedure for analysis of potential failure modes within a system for classification by severity or determination of the effect of failures on the system. Its functions as following:

- Recognize and evaluate the potential failure of a product/process and the effects of the failure.
- Identify actions that could eliminate or reduce the chance of the potential failure occurring.
- Complementary to the process of defining what a design or process must do to satisfy the customers.

2.3 FTA

FTA is

- an analysis method with Fault Tree
- an analysis to display the relationship of different fault
- a deductive method from top-level event not inductive

3 Put Linux in some restrictive conditions

For Linux is very complex and large, we need put it in some specific conditions: specific applications (including specific configuration, arguments and environments) with specific kernel executing paths in kernel internal. For describing it more conveniently, the basic GNU/Linux architecture is shown in Figure1.

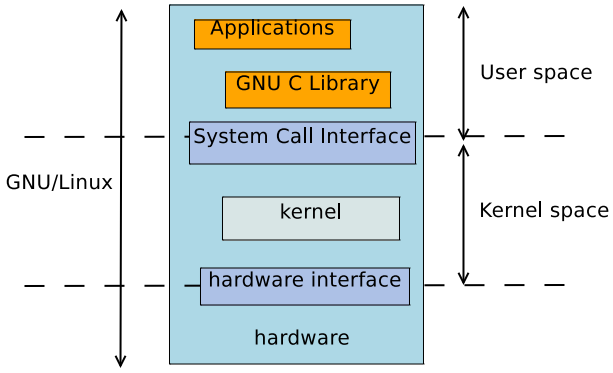


FIGURE 1: *Basic GNU/Linux architecture*

When running the specific applications, which with relative arguments and configuration in a given environment, the system calls triggered directly by the applications or indirectly via libs will be fixed. And the kernel functions called by the system calls will also be fixed, and the same as the hardware-interfaces. The system calls triggered by the applications can be traced by strace, and the calling paths from the system calls to the hardware-interfaces can be traced by some kernel tracing tools such as kft[5], ftrace[6]. And if we map the system calls and kernel functions to the relative source code files via some tools like ctags, we can analyze them via some formal tools. And further, if we running the application enough times and profile the kernel in source code line level, we will trace the most-often executed lines or blocks(several lines together) of the source code in the kernel functions, and then the hotspots to evaluate will be shorted. There is really such a tool for profiling kernel, it is kgcov[7], and the user-space tool gcov[8] can help us map the profiling result to the source code files for further analyzing like the autoctags does. The tools and their functions as following:

- autostrace: find out the system calls called by the applications
- autokft: trace all of kernel functions called by the system calls
- autoctags: find out the source code files who defines the relative kernel functions in the path of the Linux kernel directory
- autokgcov: test the coverage of the code in the kernel functions(kernel space)
- autogcov: test the coverage of the code in the kernel functions(user space)

4 Come to whole sil4linux system

The formal two parts have introduced the principle of our sil4linux project, now a whole sil4linux will be discussed, including the design architecture, implementation and usage.

4.1 System architecture

The whole system architecture is given in figure 2 to emphasize the main components in the system.

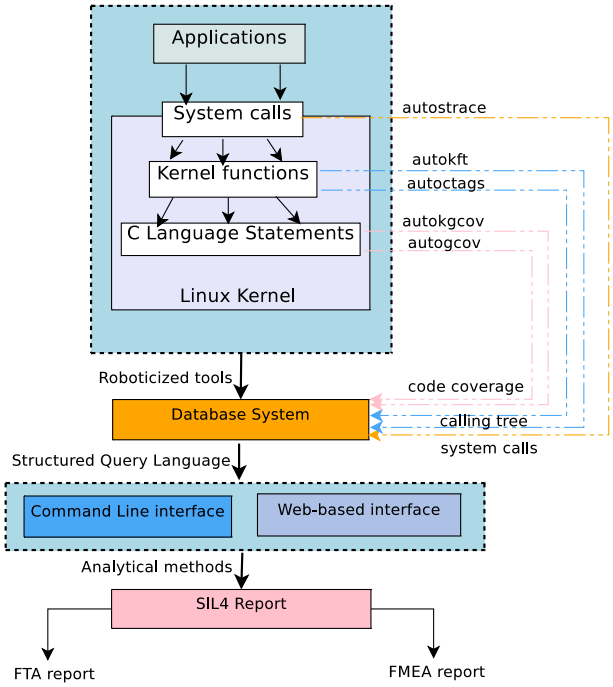


FIGURE 2: *sil4linux architecture*

In the above architecture,there are four parts:

- Roboticized tools
- Database system
- Interface
- Result report

The roboticized tools are autostrace, autokft, autoctags, autogcov, autokgcov. All the tools and their functions have presented in section 3.
As the goal of the SIL4Linux project is trying to provide information about the possibility of using Linux for SIL4 application. But as we know, Linux kernel is too complex to be analyzed directly. So we should try to find out a representative sub-collection of it and then analyze it. For the whole structure of Linux like a "big" tree, and it's too big to be analyzed directly. However, if some methods can find out the representative trees appear frequently (perhaps relative with the special application area), then just analyze some of these trees. So if using some special test-suites which cover most of the system calls, and do enough numbers of tracing with strace, kft and gcov as many as possible, then may get the representative trees. And there is really such a test-suites named POSIX test-suite. The Open POSIX Test Suite is an open source test suite with the goal of performing conformance, functional, stress, and performance testing of the functions described in the IEEE Std 1003.1-2001 System Interfaces specification[9].

4.2 System design

4.2.1 Database design

To save the result of all the tools, by selecting a database management system(this project use PostgreSQL), and using SQL to do some relative statistic and analyzing.
what need save including the experiment environment, the system calls, the kernel functions, the calling trees of the system calls, the program parts(code blocks) of the kernel functions, and some relative information, such as the the files who define the relative functions. So we design a database named sil4 with different tables to store the above information.

4.2.2 Interfaces design

For simplifying the operation of the SIL evaluating procedure, two different interfaces have been designed, one is the command line interface, another is the web-based interface.

1. CLI interface. In CLI interface, we have two main script kft_query and gcov_query. kft_query is a shell script which designed to query the kft relative tables in the sil4 database, it including

   - list all of the system calls
   - list the the source code files(in Linux kernel) who define the system calls
   - list the calling trees called by the system calls

   gcov_query is a tool for searching the Gcov relative information, mainly used to search three different types of source code of the kernel functions.

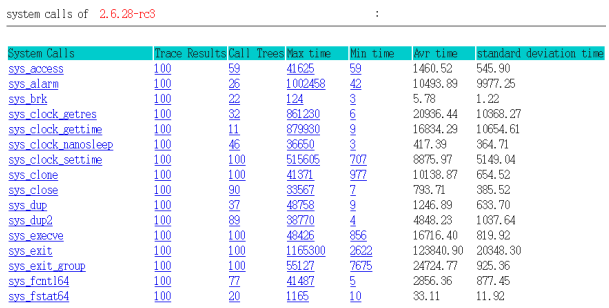
2. Web Interface. We can access the data which selected from sil4 project via web-based interface by provide a user name and password to login.

4.3 Implementation

The implementation includes several parts:

- the implementation of automatic tools in the system architectures: autostrace, autokft, autoctags, autokgcov, autogcov. These have been discussed in the above parts.
- the implementation of the database system: here using PostgreSQL DBMS to manage the database. PostgreSQL[10] is an object-relational database system that has the features of traditional commercial database systems with enhancements to be found in next-generation DBMS systems. And PostgreSQL is free and the complete source code is available.
- the implementation of the interface which have shown in the above part.

And the following picture will show some results of our sil4linux project.

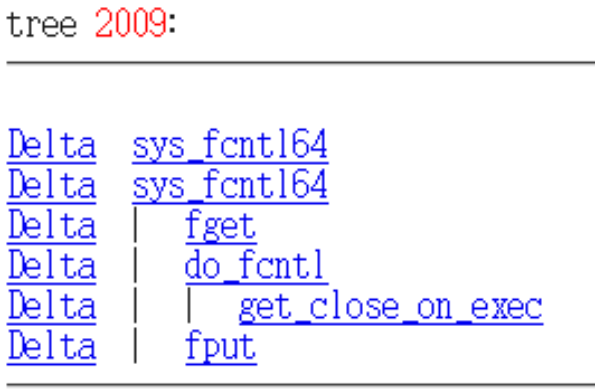


system calls of 2.6.28-rc3 :

System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time
sys_access	100	59	41625	59	1460.52	545.90
sys_alarm	100	26	1002458	42	10493.89	9977.25
sys_brk	100	22	124	3	5.78	1.22
sys_clock_getres	100	32	861230	6	20936.44	10368.27
sys_clock_gettime	100	11	879930	9	16834.29	10654.61
sys_clock_nanosleep	100	46	36650	3	417.39	364.71
sys_clock_settime	100	100	515605	707	8875.97	5149.04
sys_clone	100	100	41371	977	10138.87	654.52
sys_close	100	90	33567	7	793.71	385.52
sys_dup	100	37	48758	9	1246.89	633.70
sys_dup2	100	89	38770	4	4848.23	1037.64
sys_execve	100	100	48426	856	16716.40	819.92
sys_exit	100	100	1165300	2622	123840.90	20348.30
sys_exit_group	100	100	55127	7675	24724.77	925.36
sys_fcntl64	100	77	41497	5	2856.36	877.45
sys_fstat64	100	20	1165	10	33.11	11.92

FIGURE 3: *a part of system calls of Linux kernel 2.6.28-r3*

From the above figure, you can get information about the system calls, trace result, call trees, Max time, Min time, Average time and standard deviation time. When click the call trees of sys_geteuid32, the information is shown in the following figure.



tree 2009:

Delta sys_fcntl64
Delta sys_fcntl64
Delta | fget
Delta | do_fcntl
Delta | | get_close_on_exec
Delta | fput

FIGURE 4: *trees of sys_geteuid32*

4.4 Usage

Before evaluating the kernel whether satisfying SIL 4, there are some procedure need do at first. First, need to install sil4linux system and basic tools, such as gcc, python, PostgreSQL, apache, ctags, posix-testsuite and so on. And some special tools such as FMEA, FTA, gcov, kft, strace and so on. Second, we need to run autostrace, autokft, autoctags and autogcov. Then import the result to the database. Because we have create tables in the database, we only run the relative script to import different result files to the database. Third, we need script to access all results via command line interface and web-based interface. Finally, we can analyze results in the formal method.

5 The extra functions sil4linux can do

The extra functions the sil4linux can do including:

- it can help for Linux kernel study and development like LXR[11] does.
- guides for Linux kernel performance tunning in some specific application, for example, running Baidu or Google Spider in Linux, and run sil4linux for it, and then tunning Linux kernel for running Baidu & Google Spider quicker.
- some other potential database dig, such as compare the different system features when enabling/disabling kernel configuration, when migrating Linux kernel from a version to another and so forth.

6 Conclusions

In the modern world we live nowadays, mankind depends on a variety of technical,highly sophisticated systems. And the operating system Linux and its kernel is well known and used in thousands of desktop computers, servers and embedded systems all over the world. The thesis gave a possibility of assessing Linux satisfying SIL 4 applications in some restrictive conditions. It would be useful to provide a method that it is possible for Linux would meet a SIL 4 integrity requirement. And SIL4Linux is a project for finding out some available informations to use Linux for SIL4 applications under some restrictive conditions. At last, a archetypal method based on some formal methods, like FMEA, FTA have been designed.

7 Acknowledge

At first, it is very grateful of M. Sc Wu Zhangjin for his great contribution to sil4linux. His work makes all of us know more about SIL 4, KFT, GCOV, Linux kernel, shell script, debugging tools. Thanks a lot to Prof. Nicholas Mc Guire, Dr. Zhou Qingguo and all the other people of DSLab for their discussion on SIL 4 questions. This work was supported by Siemens.

8 Acronyms

- CLI - Command Line Interface
- COTS - Commercial Off The Shelf
- DBMS - DataBase Management System
- FMEA - Failure Modes and Effects Analysis
- FTA - Fault Tree Analysis
- GCOV - test COVerage program for Gnu cc
- IEC - International Electro-technical Commission
- ISA - Instrument Society of America
- KFT - Kernel Function Trace
- LXR - Linux Cross Reference
- OSS - Open Source Software
- PFD - Probability of Failure of Demand
- RRF - Risk Reduction Factor
- SIL - Safety Integrity Level
- SIF - Safety Instrumented Function
- SIS - Safety Instrumented Systems
- SQL - Structure Query Language

References

[1] intr,H.R. Pierce, *Preliminary Assessment of Linux for safety related systems*,Research Report 011, 2002, HSE, ISBN: 0 7176 2538 9.

[2] Nicholas Mc Guire,*Linux for Safety Critical Systems in IEC61508 Context*,2007

[3] http://www.gmsystemsgroup.com/sil/sil_faqs.html

[4] http://en.wikipedia.org /wiki/Failure_mode_and_effects_analysis

[5] http://tree.celinuxforum.org/CelfPubWiki /KernelFunctionTrace

[6] http://linux.die.net/man/1/ftrace

[7] http://oss.lzu.edu.cn/blog/article.php?tid_2028.html

[8] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[9] http://posixtest.sourceforge.net/

[10] http://en.wikipedia.org/wiki/PostgreSQL

[11] http://lxr.linux.no/