-浅谈嵌入式系统的安全 -

周庆国

< zhouqg@lzu.edu.cn >

兰州大学分布式与嵌入式系统实验室

Distributed & Embedded System Lab

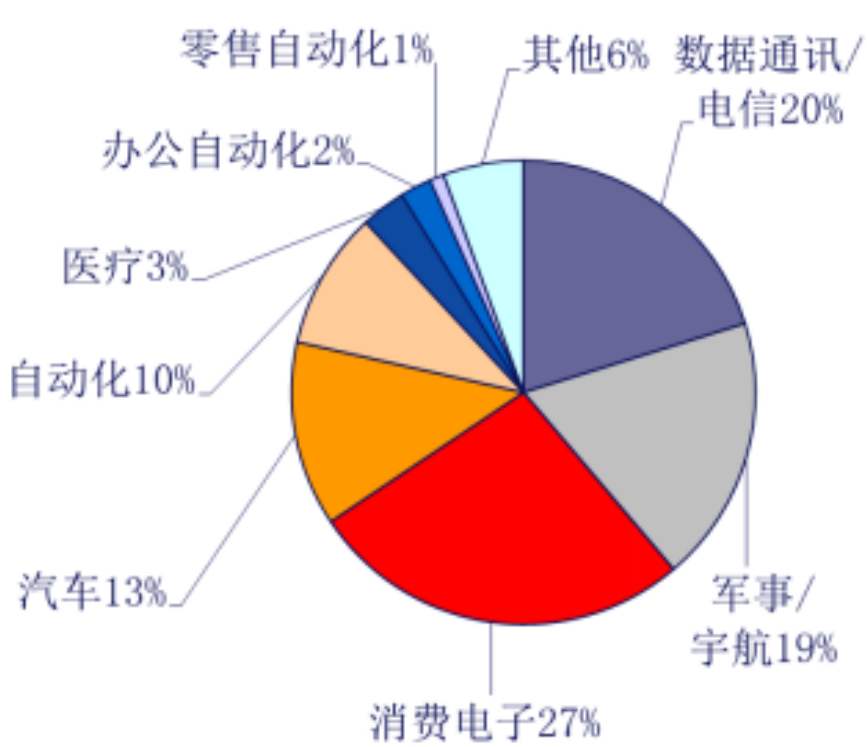
http://dslab.lzu.edu.cn

2015-11-12

目录

背景介绍

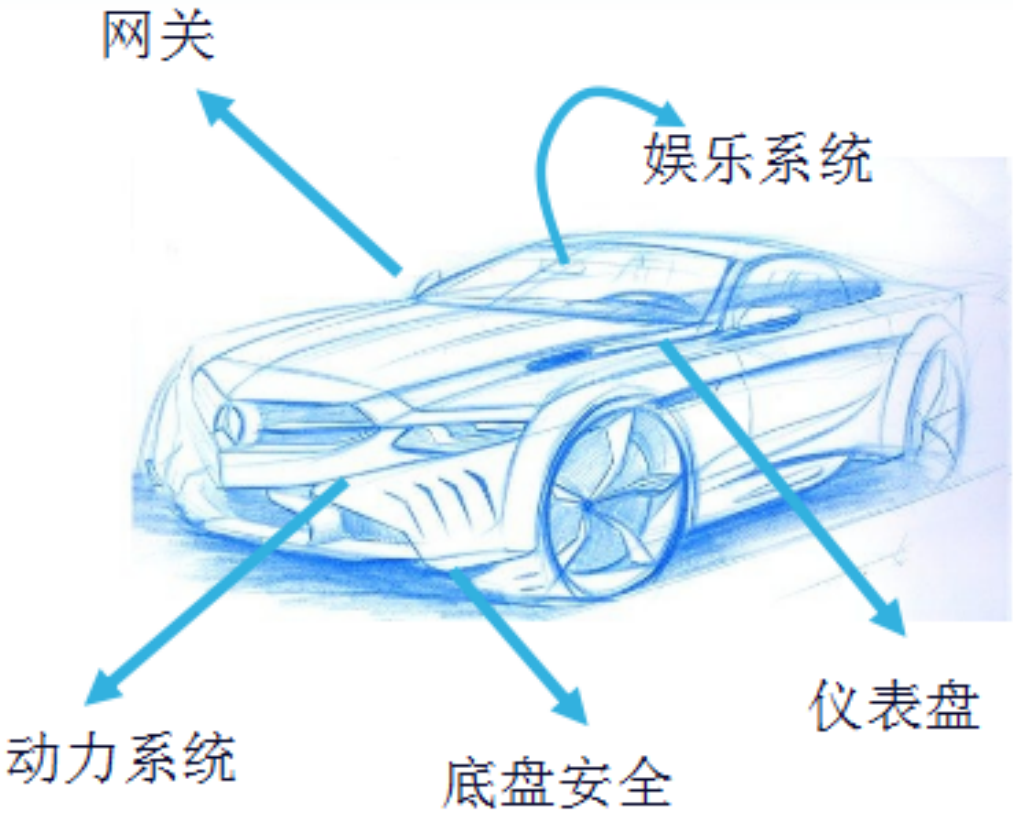
嵌入式软件分布



新趋势：

- 电信、军事、消费电子、汽车比重增大。
- 随着IOT、云计算等发展迅速，消费电子、智能家电等用户逐年增加。
- 汽车、宇航等领域关键安全系统引起人们极大重视。

背景介绍



汽车里典型的嵌入式系统：

- 前置信息娱乐系统
- 防抱死制动系统
- 发动机控制单元
- 数字仪表
- 其他电子子系统
- ……

这些都是**嵌入式系统**

背景介绍

- 嵌入式系统作为安全关键系统被广泛应用到各生产行业中，但是近年来随着一些安全事故的发生，人们越来越关注其安全问题。
- Eg：2011 年，温州高铁事故；

背景介绍

- 系统越复杂，可靠性越低。
  - 最低端的汽车至少有几十个微处理器；
  - 嵌入式代码软件很容易超过 1 亿行；
  - 代码越来越复杂，系统安全性越来越受关注；
  - Linux 内核平均每天增加超过 12000 行代码；
  - 在 Linxu-2.6.30 版本中已经超过 1100 万行代码。

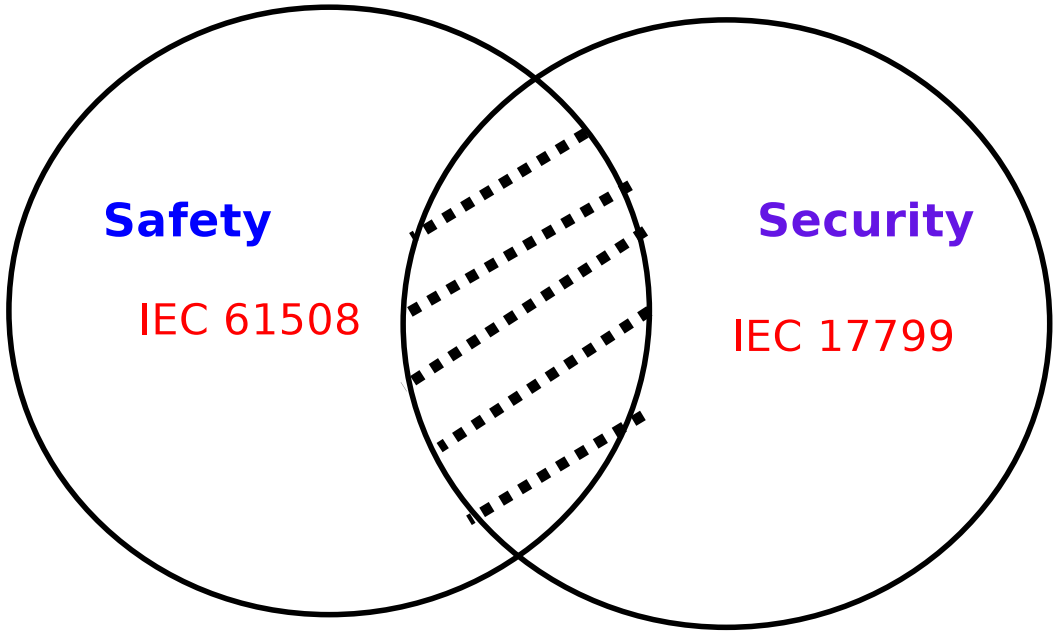
概念简介

- **什么是安全 (Safety or Security)？** ”安全”对应的词汇有 Safety 和 Security，两者存在很大区别：
  - Security ——IEC 17799 标准[1]将 Security 定义为保护外来伤害的能力，适用于脆弱和有价值的资产，如人、组织、住房、国家等；
  - Safety ——IEC 61508 标准[2]将 Safety 定义为：将能够造成人员伤亡或财产损失以及严重破坏环境的危险降低到可以接受范围;

[1] 信息安全管理实施细则（Code of Practice for Information Security Management）

[2] 全名为 “E/E/PE 安全相关系统的功能性安全标准”，是一套面向功能性安全（Functional Safety）的通用标准。已被多国认可，如我国的对应标准 GB/T 20438

概念简介



- 所以 Security 主要是针对外来伤害，如网络攻击；而 Safety 主要是保证危害是可控的从而不会对人们的财产和生命造成威胁 (关注方向)。

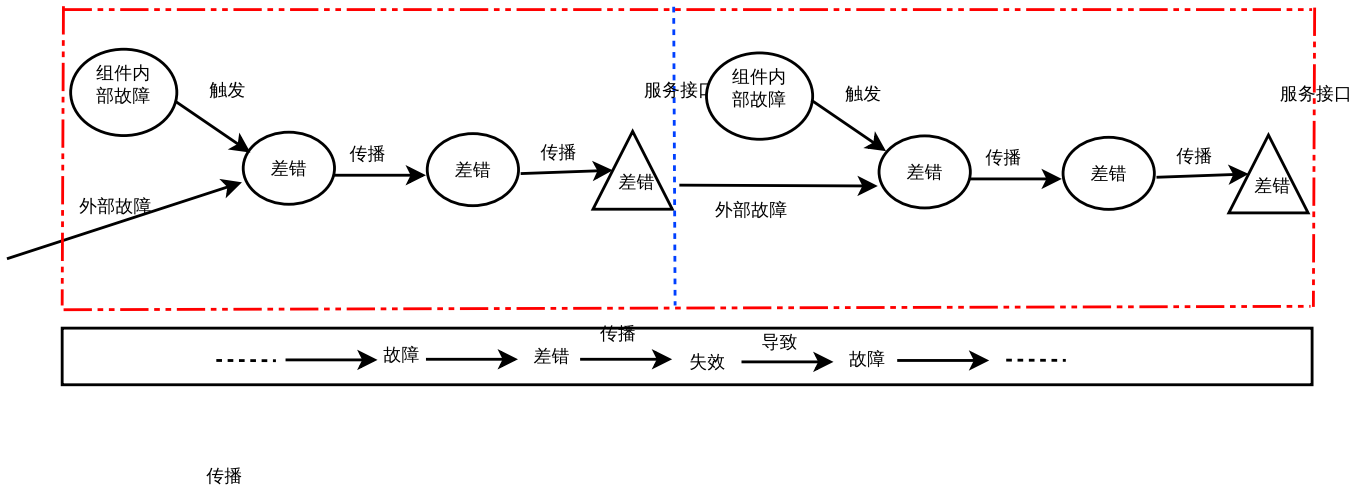
概念简介

- **Safety 的关键因素**
  - 可预测性——对系统中可能出现的风险都能考虑到。
  - 确定性——可识别风险在控制范围内。
- **安全领域的故障分类**
  - 随机性故障——随机发生的、不可控，可能是由于硬件老化产生，或者受外在因素影响（电磁场变化、宇宙射线等），一般可以通过软硬件冗余解决。
  - 系统性故障——永久性故障，一般由某一固定因素引起，以特定形式出现的故障。只能通过修改设计、制造工艺、操作程序或其它关联因素消除。

概念简介

- 故障（Fault）——导致失效和差错的原因。
- 差错（Error）——系统的功能偏离预定值或状态，一般指组件或系统的内部状态。**当差错传播影响到系统或组件的对外接口时，就有可能转变为失效**
- 失效（Failure）——系统或组件不再具有提供预定功能的能力，或提供的功能与预定功能偏离；

概念简介



- 当满足一定条件后，组件（或系统）内部或外部的故障被触发，转换为差错，经过一系列过程转化为不同的差错状态，到达组件（或系统）的接口处，转化为失效。**前者的失效在一定程度上是后者（系统或组件）的故障**

概念简介

- **安全关键系统概念** ——这样一类系统：一旦系统失效，则会导致人员或财产损失、或对环境造成严重破坏。研究这种系统的目的是通过降低风险发生的概率、减少事故造成的灾难，使得风险达到一个可接受的范围。
- **安全完整性等级** ——在控制系统的安全性和可靠性评估中，有些标准（eg：ISA 的 S84.01[3]、IEC61508 等）提供了一些性能指标，通过这些指标可以比较安全性与可靠性的计算结果，以及系统如何设计才能最大限度的提高安全性和可靠性。

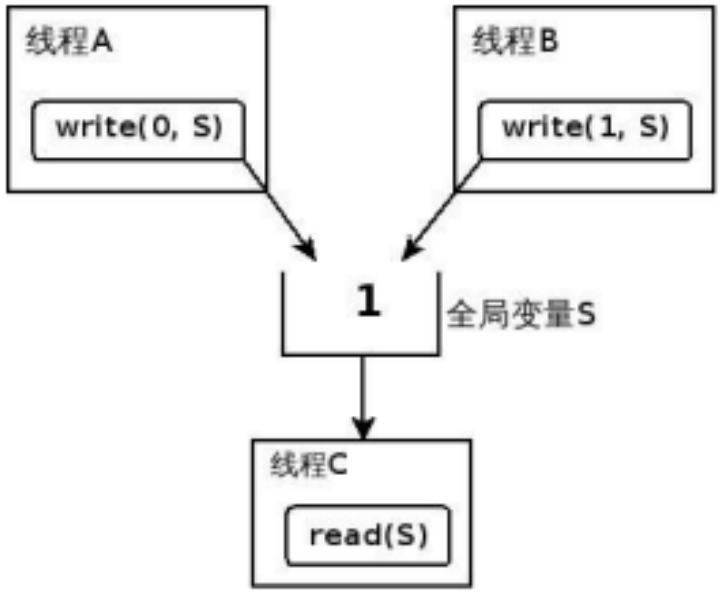
[3] 过程工业安全仪表的应用

概念简介

- 安全完整性等级——IEC61508 标准

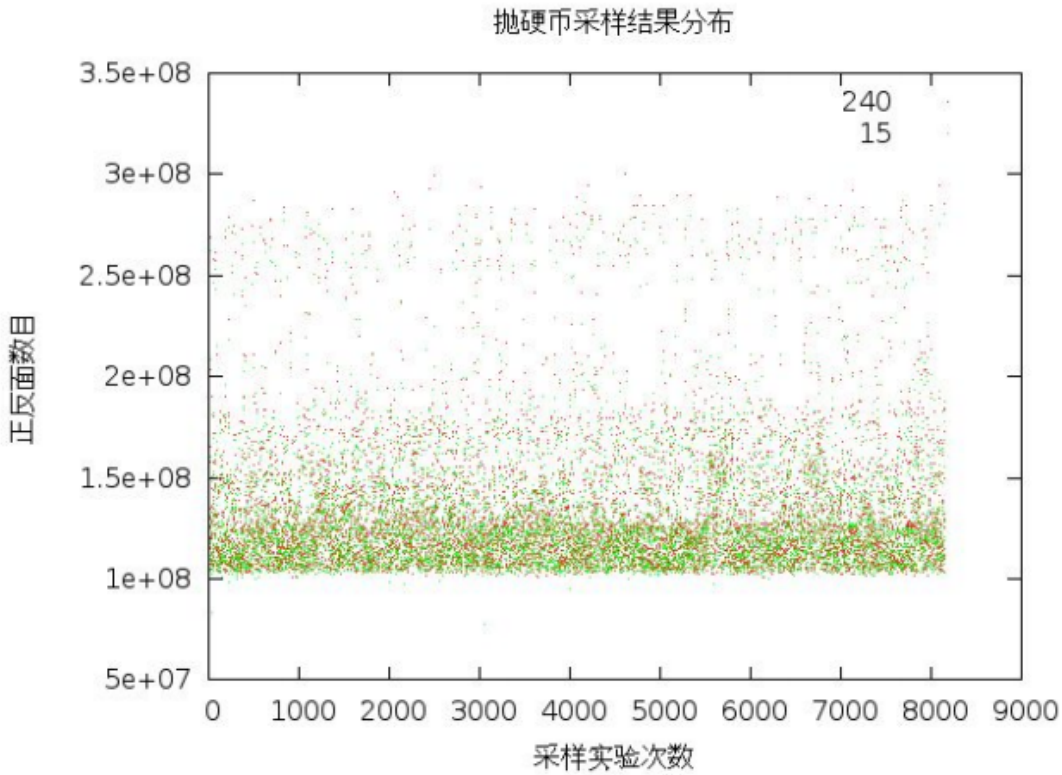
安全完整性等级 SIL	需求失效概率	风险降低因子
4	<0.0001	>10000
3	$0.001 \sim 0.0001$	$1000 \sim 10000$
2	$0.01 \sim 0.001$	$100 \sim 1000$
1	$0.1 \sim 0.01$	$10 \sim 100$

Case Study —抛硬币模型



- S: 系统中的唯一全局变量；线程 A、B 共享竞争共享资源 S，且 A、B 之间不存在任何同步机制；线程 C 对全局变量 S 采样。
- 硬件平台：Intel(R) Core(TM)2 Quad Procesor Q6600

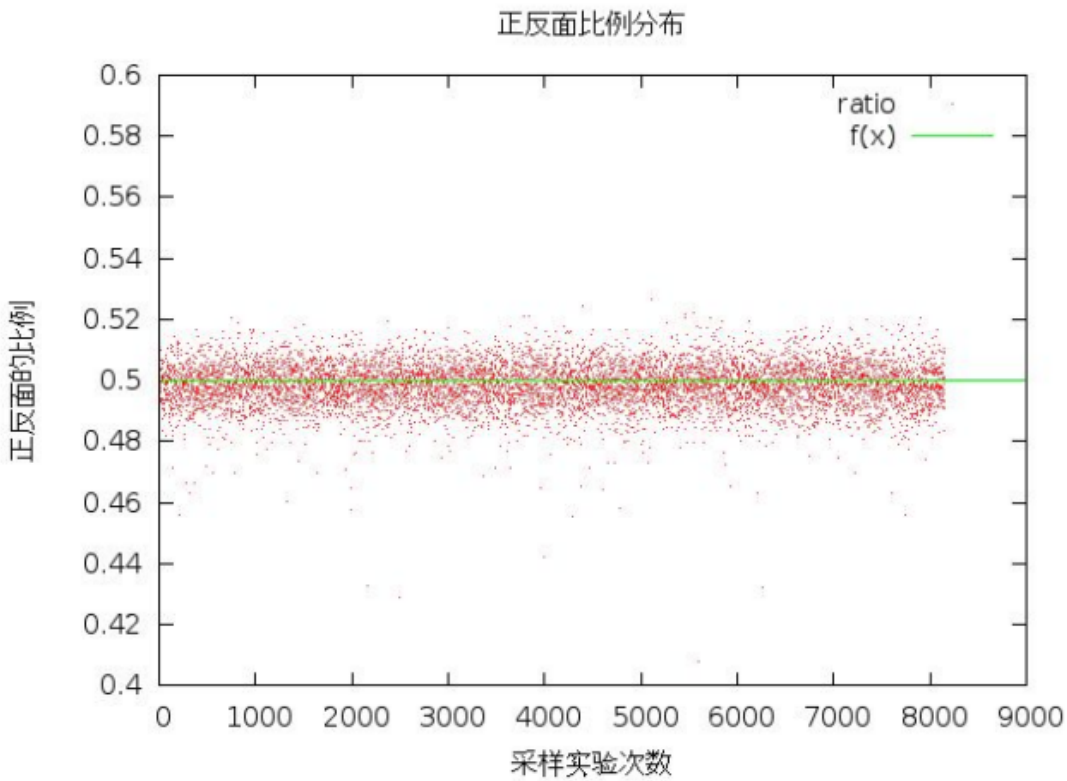
Case Study —抛硬币模型



抛硬币采样结果:0、1 出现的几率毫无规律

Case Study —抛硬币模型

硬币正反面比例



系统随机性产生因素

- 访问内存的操作被 DMA 阻塞
- 未命中 TLB 项
- 未命中 Cache 行
- 指令乱序执行
- 中断发生
- 多核处理器的 IPI 机制（inter-processor interrupt）
- ……

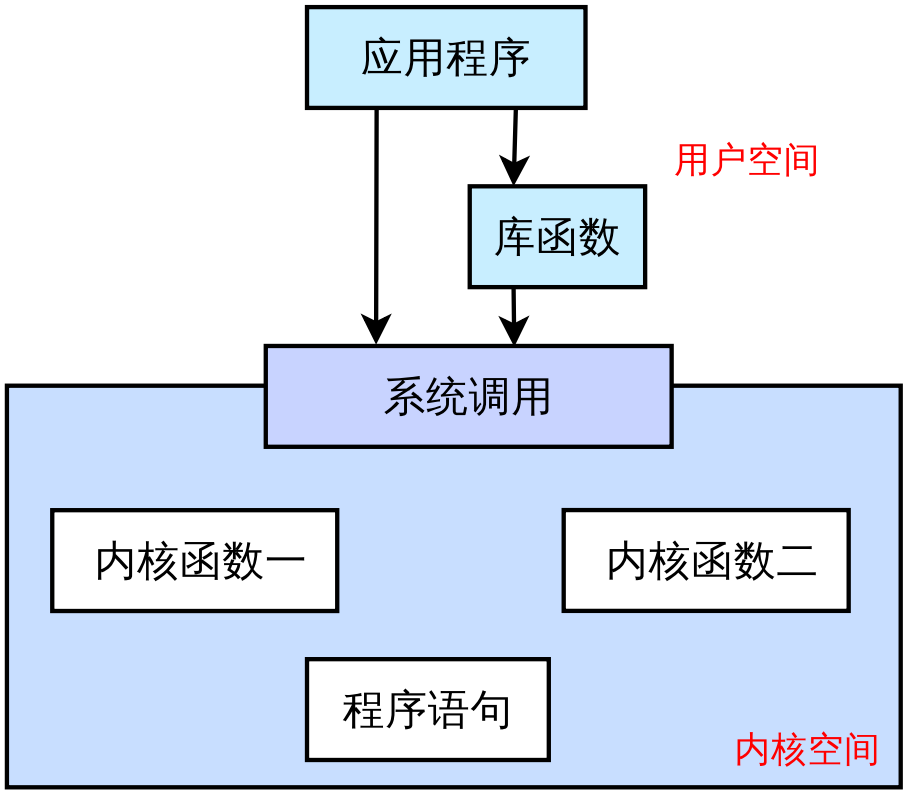
系统随机性带来的安全问题

- 随机性故障发生是随机的，不可预测的，这种故障会影响到系统中的所有实例。
- 这种问题普遍存在于嵌入式系统中。

嵌入式系统——GUN/Linux

- Linux 操作系统的配置选项多达上千项，具有很强的可定制性；
- 嵌入式 Linux 是以 Linux 为基础的嵌入式作业系统，它被广泛应用在移动电话、个人数字助理 (PDA)、媒体播放器、消费性电子产品以及航空航天等领域中；
- 嵌入式 linux 是将日益流行的 Linux 操作系统进行裁剪修改，使之能在嵌入式计算机系统上运行的一种操作系统。

Linux 系统架构



GNU/Linux == Safety ?!

- GNU/Linux 没有以 IEC 61508 为开发基础，也没有贯彻安全开发生命周期
- 但是其有着一些优势使其有着使用在安全领域的潜力，如：可追踪性、文档、高级的项目管理、测试和验证 etc.
- 目前国内外，没有任何研究团队或个人能够给出 Linux 系统安全验证的模型。

嵌入式 Linux 系统安全验证面临的问题

- **Linux 复杂度高、规模大**
- **系统行为多样性无法量化**

Case Study——Sil4Linux Project

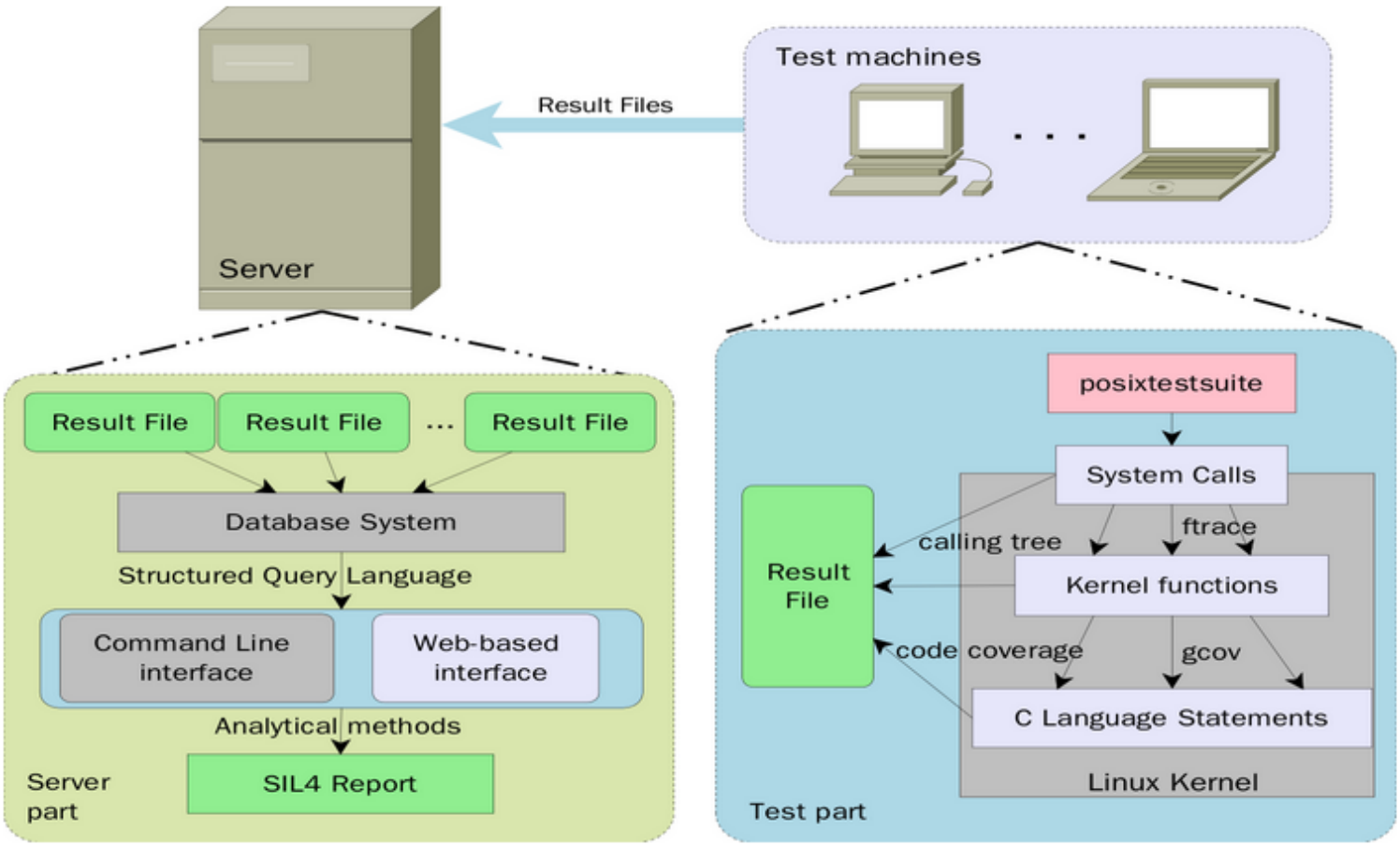
- **什么是 SIL ？**
  - SIL 即 Safety Interity Level，是 IEC 61508 中提出的一个概念用于描述**“安全相关系统实现需求目标的置信度”**
  - “安全完整性”（Safety Interity），IEC 61508 中定义为：**“在一定的时间周期内，一定的条件下，安全相关系统实现其预定的安全功能的可能性（概率）”**
- **Sil4Linux 项目的目的**
  - 试图分析 Linux 内核并证明 Linux 系统在某些条件下满足 SIL4；

    `http://sil4linux.dslab.lzu.edu.cn/`

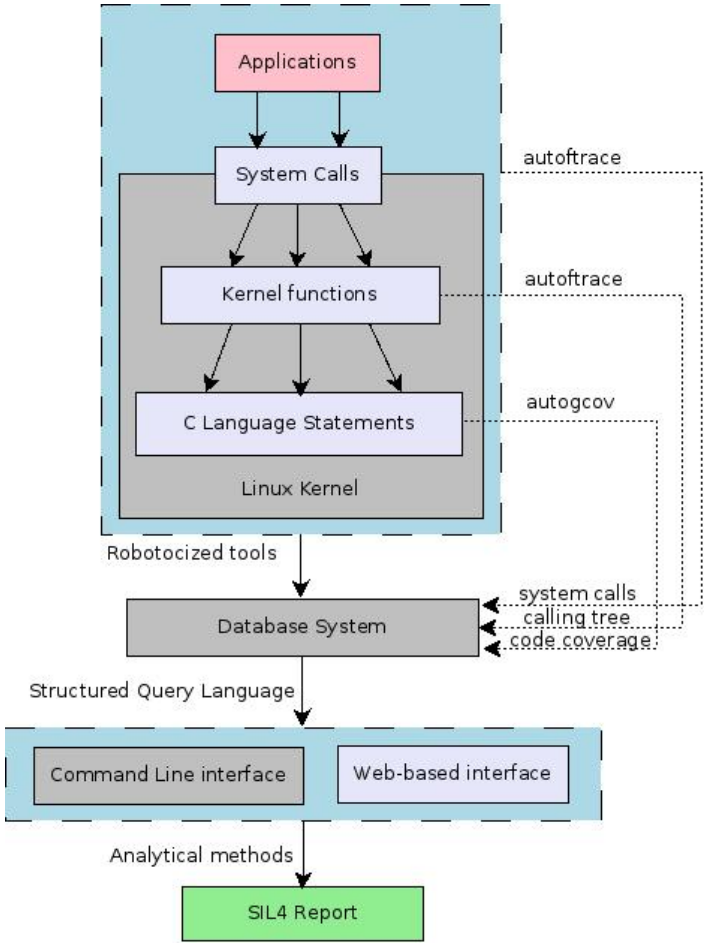
嵌入式 Linux 系统安全验证问题解决办法

- 找到一个切入点、记录系统重要的行为信息，获得一个有代表性的、可以验证的子集
- **系统调用**（system calls）是一个很好的切入点，因为所有的应用都需要通过系统调用进入内核空间，继而调用内核函数，在细节就涉及到内核函数的每行代码。
- 一个系统行为追踪 100 次，以便量化以及对比不同系统的行为比例。

Sil4Linux 总体架构图



Sil4Linux 总体架构图



Sil4Linux Project

linux kernel 2.6.23 posix testsuit systemcall map Timer frequency 100Hz
linux kernel 2.6.23 posix testsuit systemcall map Timer frequency 250Hz
linux kernel 2.6.23 posix testsuit systemcall map Timer frequency 1000Hz

linux kernel 2.6.24 posix testsuit systemcall map none
linux kernel 2.6.24 posix testsuit systemcall map voluntary preempt
linux kernel 2.6.24 posix testsuit systemcall map preempt
linux kernel 2.6.24.7-rt17 posix testsuit systemcall map none
linux kernel 2.6.24.7-rt17 posix testsuit systemcall map voluntary preempt
linux kernel 2.6.24.7-rt17 posix testsuit systemcall map preempt

linux kernel 2.6.25 posix testsuit systemcall map
linux kernel 2.6.26 posix testsuit systemcall map
linux kernel 2.6.28-rc2 posix testsuit systemcall map
linux kernel 2.6.28-rc3 posix testsuit systemcall map
linux kernel 2.6.29 posix testsuit systemcall map

system calls of 2.6.23_100Hz :

System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time	posixtestsuite file
sys_access	100	86	8640	32	464.53	86.37	posix files
sys_alarm	100	4	95	22	24.43	0.71	posix files
sys_brk	100	5	11943	1	149.80	121.81	posix files
sys_clock_getres	100	5	31	3	4.16	0.32	posix files
sys_clock_gettime	100	3	17	4	5.88	0.12	posix files
sys_clock_nanosleep	100	48	9320	2	1012.75	247.59	posix files
sys_clock_settime	100	96	155446	70	3395.66	2058.68	posix files
sys_clone	100	93	8560	323	1096.41	83.05	posix files
sys_close	100	71	4081	4	186.51	55.07	posix files
sys_dup	100	1	7	6	6.46	0.05	posix files
sys_dup2	100	84	10864	4	1102.35	270.03	posix files

Sil4Linux Project

- Kernel Version Of: trace-3.14.8-1000-preempt-python
- Kernel Version Of: trace-3.14.8-1000-voluntary-python
- Kernel Version Of: trace-3.14.34-1000-preempt-python
- Kernel Version Of: trace-3.18.8-1000-preempt-python

System calls of trace-3.14.8-1000-preempt-python:

System Call id	System Call name	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time	posixtestsuite file
0	SyS_access	100	2	183	53	57.187	161.377	posix files
1	SyS_alarm	100	1	164	23	35.042	605.339	posix files
2	SyS_brk	100	1	2	1	1.462	0.017	posix files
3	SyS_clock_getres	100	1	7	6	6.013	0.046	posix files
4	SyS_clock_gettime	100	1	22	20	20.709	0.109	posix files
5	SyS_clock_nanosleep	100	6	2810	1657	2044.792	68796.808	posix files
6	SyS_clock_settime	100	2	187	4	99.622	3685.516	posix files
7	SyS_clone	100	11	429	127	136.474	864.267	posix files
8	SyS_close	100	1	9	8	8.41	0.007	posix files
9	SyS_dup	100	1	12	11	11.282	0.066	posix files
10	SyS_dup2	100	1	17	4	14.782	3.88	posix files
11	SyS_execve	100	100	9372	75	8203.026	1440117.514	posix files
12	SyS_exit	100	2	0	0	0.0	0.0	posix files
13	SyS_exit_group	100	3	0	0	0.0	0.0	posix files
14	SyS_fcntl	100	1	4	3	3.525	0.016	posix files
15	SyS_fsync	100	96	52052	488	36042.114	170925139.822	posix files
16	SyS_ftruncate	100	1	90	24	44.197	736.898	posix files
17	SyS_futex	100	1	160	11	23.016	511.462	posix files
18	SyS_getdents	100	40	735	187	273.598	17197.391	posix files
19	SyS_getitimer	100	2	49	11	11.706	15.929	posix files
20	SyS_getpgid	100	1	35	9	10.269	6.912	posix files
21	SyS_getrlimit	100	1	182	6	20.127	489.139	posix files
22	SyS_ioctl	100	1	74	22	55.803	438.974	posix files
23	SyS_kill	100	5	283	66	77.774	1062.608	posix files
24	SyS_link	100	9	1100	242	548.605	81368.145	posix files

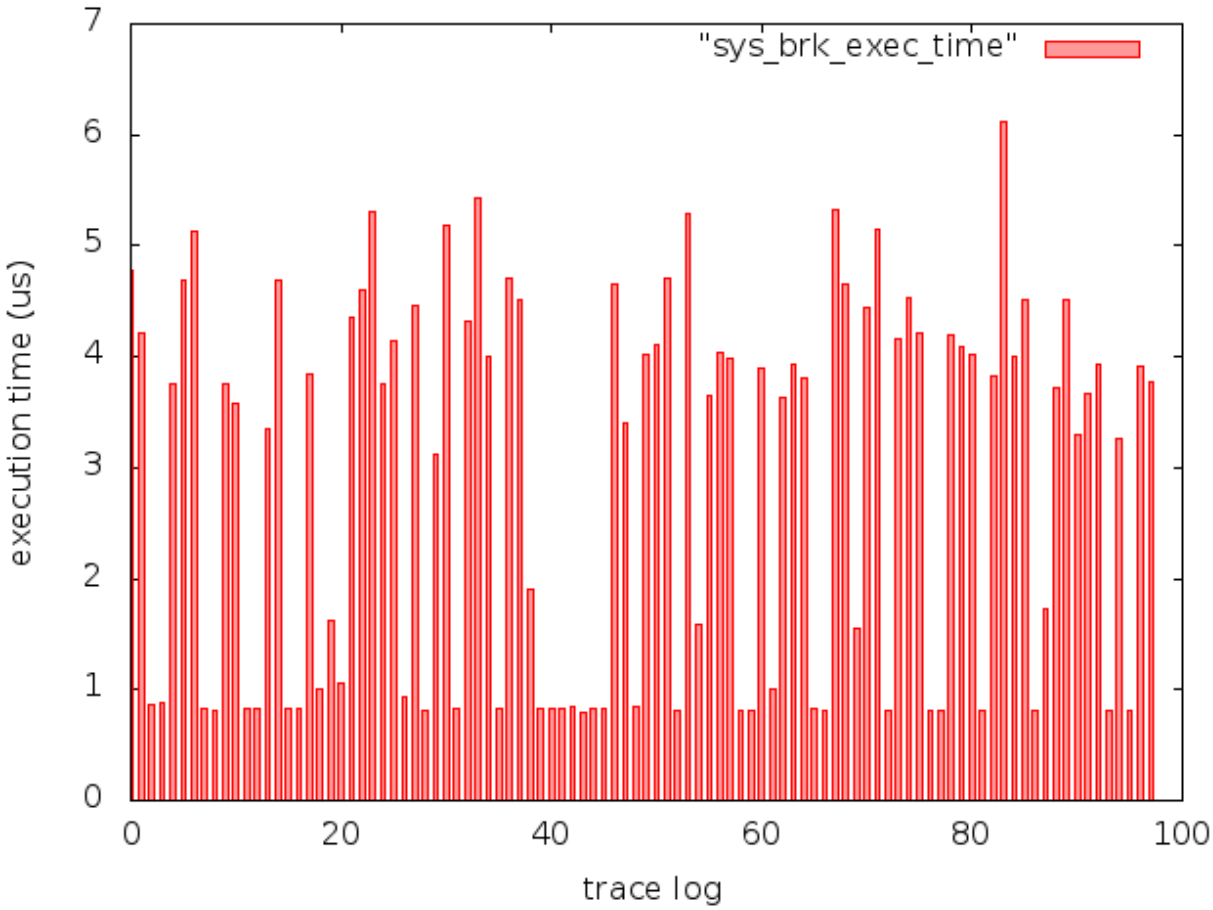
Sil4Linux 执行流程

- 使用 posixtesuite 测试集，其中部分测试会调用系统调用以获取内核空间提供的服务或者资源；
- 使用追踪器对测试过程进行追踪，获取有系统调用的追踪结果。对每个系统调用追踪 100 次。追踪结果可以用来对常用路径或非常用路径在一定程度上得到一个量化；
- 对于追踪结果进行分析，筛选系统调用以及相关的内核函数，使用代码覆盖工具对其进行分析，从而可以结合追踪结果分析内核行为。
- 服务器端程序负责解析测试结果；将解析的数据结构化地存储到数据库；并通过 web web 接口展示测试结果。

实验工具简介

- POSIXtesuit——:POSIX 是测试程序的集合，其中每一个测试程序的执行都会触发一些系统调用，从而为以系统调用为突破口对 Linux 操作系统内部执行动态的研究提供了保证。
- Ftrace——是 Linux 内核中内嵌的一种追踪工具，2.627 版本开始新增加入内核，主要用途是追踪 Linux 内核运行时的动态行为。
- Gcov——Gcov 是代码覆盖率测试工具，主要用来计算 C / C++ 代码执行语句或分支的覆盖率，通过计算代码中的未被执行的部分做冗余代码检测。

执行时间的不确定性


"sys_brk_exec_time"
execution time (us)
trace log
0
1
2
3
4
5
6
7
0
20
40
60
80
100

sy_brk 作示例，为什么选择它？

```
SyS_brk() {
    down_write() {
        _cond_resched();
    }
    up_write();
}
```

- 因为这个系统调用的执行路径非常简单,100 条追踪结果都是走的同一条执行路径;
- sy_brk 在 100 组中的结果都显示在同一路径下，这也正是前文中提到的“常用路径”。

执行时间的不确定性

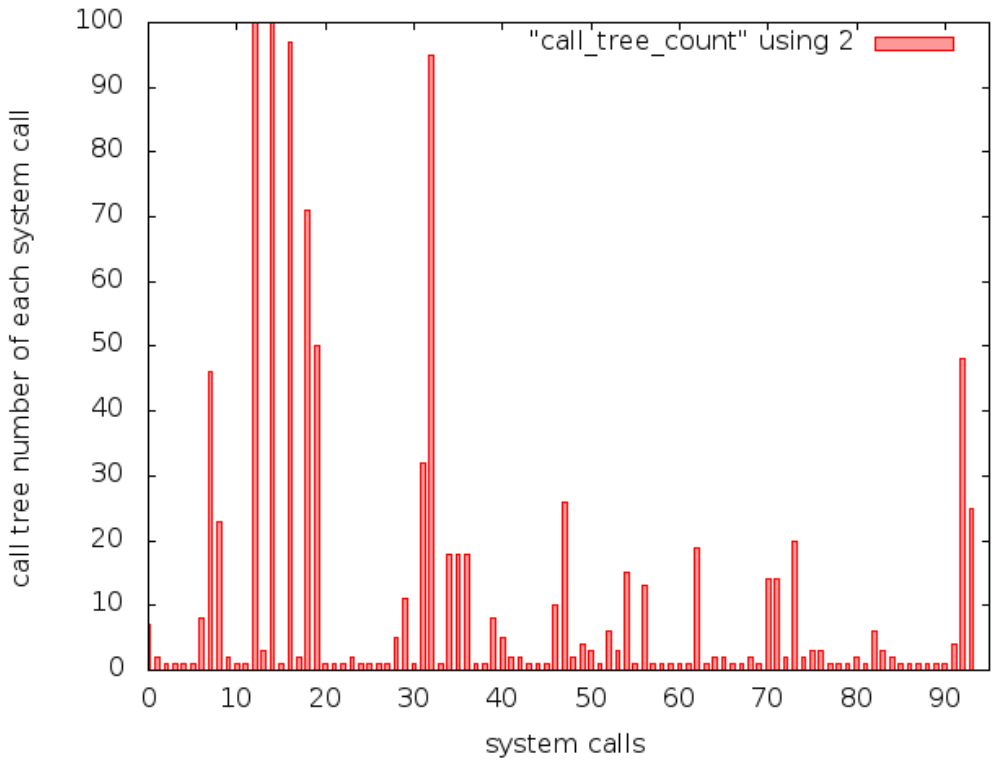
- Kernel Version : Linux-3.18.8

System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time
sys_access	100	75	11167	34	515.08	158.36
sys_alarm	100	11	84	23	27.73	1.43
sys_brk	100	4	10	2	2.96	0.12
sys_clock_getres	100	7	13	4	4.71	0.20

- Kernel Version : Linux-4.1.3

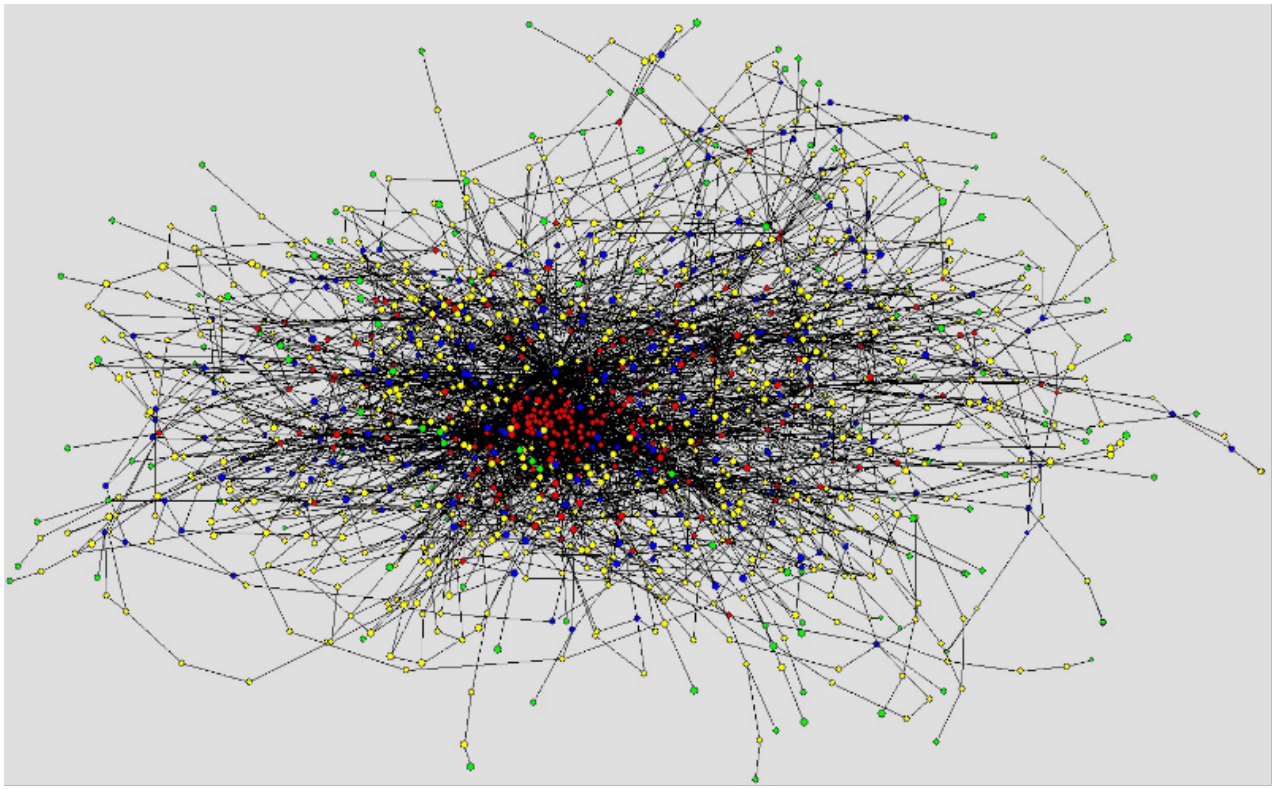
System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time
sys_access	100	73	36347	54	2133.07	574.65
sys_alarm	100	15	234	40	51.72	3.17
sys_brk	100	17	26	3	4.25	0.32
sys_clock_getres	100	9	223704	5	2243.21	2225.76

执行路径的不确定性



- 有一部分系统调用在 100 次追踪结果中只有一个调用树，也即都运行在常用路径下；而更多的系统调用有着不止一个路径（具体哪个是常用路径则需要对比具调用树的数量了）。

执行路径的不确定性



以 SIL4Project 追踪到的内核函数为节点，以函数之间的调用关系为边，组成的 3D 网络图。

执行路径的不确定性

- Kernel Version : Linux-3.18.8 (Tree of sys_brk)

System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time
sys_access	100	75	11167	34	515.08	158.36
sys_alarm	100	11	84	23	27.73	1.43
sys_brk	100	4	10	2	2.96	0.12
sys_clock_getres	100	7	13	4	4.71	0.20

- 执行路径概率分布

trees of sys_brk:

all_trees 100%

1326 (97) 97% 1327 (1) 1% 1328 (1) 1% 1329 (1) 1%

执行路径的不确定性

- Kernel Version : Linux-4.1.3 (Tree of sys_brk)

System Calls	Trace Results	Call Trees	Max time	Min time	Avr time	standard deviation time
sys_access	100	73	36347	54	2133.07	574.65
sys_alarm	100	15	234	40	51.72	3.17
sys_brk	100	17	26	3	4.25	0.32
sys_clock_getres	100	9	223704	5	2243.21	2225.76

- 执行路径概率分布

trees of sys_brk:

all_trees 100%

1515	(1)	1%	1516	(83)	83%	1517	(1)	1%	1518	(1)	1%
1519	(1)	1%	1520	(1)	1%	1521	(1)	1%	1522	(1)	1%
1523	(2)	2%	1524	(1)	1%	1525	(1)	1%	1526	(1)	1%
1527	(1)	1%	1528	(1)	1%	1529	(1)	1%	1530	(1)	1%
1531	(1)	1%									

常用路径和非常用路径

system call	times	tree1	tree2	tree3	tree4	tree5	tree6	tree7
sys_set	100	88	12	0	0	0	0	0
_setuid	1000	883	0	3	98	1	13	2
sys_rt	100	97	3	0	0	0	0	0
_sigreturn	1000	978	0	1	1	1	2	17
sys_clock	100	98	2	0	0	0		
_gettime	1000	991	0	6	1	2		
sys	100	100	0	0				
_setresuid	1000	998	1	1				

路径量化分析

- 常用路径所占的比重（路径对应的追踪结果数量/总的追踪次数）非常接近，如上述表格中系统调用树 1 在 100 次和 1000 次的追踪结果中所占的比例基本保持一致；
- 非常用路径的发生是不确定的，上述表中调用树 2 ～ 7 都显示出极大的不确定性，包括是否出现以及出现的几率；

总结上述结果，系统调用的执行路径是不确定的，系统调用树的数量变化较大，而且系统调用在多次的测试中出现的系统调用树及其调用树的比重都定程度上量化这些路径所占的比重（对于常用和非常用路径也可以通过量化数据得出）。

通过 SIL4Linux 验证因素

- 中断的发生是非确定的
- 不同内核版本之间的系统行为不同
- 相同内核在不同的抢占模式下的系统行为不同
- 相同内核在不同的 CPU 时钟频率下的系统行为不同
- 不同架构下的系统行为不同
- … …

广告时间

- 对安全的原则、问题和技术的一种深刻理解
- 高效地开发安全、可靠的嵌入式软件的成熟技术
- 针对系统架构、操作系统和超级管理程序、网络、存储和加密问题的研究，这些是设计安全嵌入式系统所必须考虑的问题
- 通篇使用了实用的建议和大量的案例分析

本书由重要的权威人士编写，作者拥有丰富的嵌入式安全经验，是世界上通过通用标准EAL 6+安全认证的软件产品的最初开发人员之一，而且是NSA加密认证系统的首席设计师。

对于嵌入式系统和安全专业人员，无论是新手还是资深人士，本书都是必不可少的参考书籍。

译者简介

周庆国 兰州大学教授，博士生导师。理学博士，IET会士，教育部新世纪人才基金获得者，现为兰州大学开源软件与实时系统教育部工程研究中心副主任，兰州大学分布式与嵌入式系统实验室主任。2007年10月曾受聘担任Real Time Linux Foundation主任，2011年8月曾任《International Journal of Computational Science and Engineering》（IJCSE）杂志副主编，2013年4月曾任《International Journal of Embedded System》（IJES）主编。目前主要从事安全关键系统、嵌入式系统、实时系统、虚拟化技术的研究。

姚琪 BEEcube FPGA系统工程师，有近5年的Xilinx FPGA开发和Linux驱动开发经验，曾参与过AES加密算法的FPGA实现、嵌入式操作系统移植、OpenFlow Switch开发等项目，对硬件和软件开源非常感兴趣，现专注于FPGA在SDN和SDR领域中的应用。

Thank you for your attention!
Q&A