Inherent Diversity for Fault Detection in Complex Hardware/Software Systems

Peter Okech

School of Computing & Informatics
University of Nairobi
Nairobi, Kenya

and

Systems Engineering Group
Technische Universität Dresden
Dresden, Germany

The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, June 25th 2014
Atlanta, Georgia USA
Outline

Introduction and Background

The Inherent Diversity Approach

Early Results and Discussion

Conclusion
Motivation

- Complex artifacts are usually associated with higher residual fault densities
- The safety community prefer less complex components to build safety-related systems
  - almost exclusive use of simple hardware and software platforms
- But the computing environment is rapidly changing
  - processors: sequential to superscalar to multicore
  - increasing complexity of system software/operating systems
- **Fight or embrace complexity?**
This work seeks to exploit complexity to achieve the goal of ensuring that residual faults do not lead to catastrophic failure of systems.

**Approach**: utilize the inherent non-deterministic execution in complex execution platforms to generate diversity in replicated architectures.

Objectives:

1. Investigate the level of non-determinism in the Linux kernel
2. Define a process to demonstrate system safety, based on inherent diversity
3. Quantify the potential level of coverage of inherent diversity
Background - Diversity

- General approach to deal with failure
  1. redundancy to gain fault-tolerance and
  2. fail-safety to minimize the impact of failure
- Naive redundancy of software replicates the faults in each of the copies
- Diversity in design was suggested as a solution [1, 2]
  - debates have emerged on its gains [3, 4]
  - known to be effective [5], but generating diversity is expensive
- Are there alternative methods for generating replicas?
Inherent Diversity

Figure 1: Inherent Diversity

Figure 2: The Model
Goal: Investigate path non-determinism in the Linux kernel

- Experimental Setup
  - Configured system to support kernel tracing via Ftrace
  - Traced execution of system calls, recording the kernel routines invoked by the system call

- We report results of two sets of experiments, tracing
  - a single system call, sys_open (Experiment 1)
  - multiple system calls \{read, write, lseek, fsync, nanosleep\} with a focus on sys_read (Experiment 2)

---

## Characteristics of System Call Paths

**Table 1:** Path Characteristics of *open* and *read* System Calls

<table>
<thead>
<tr>
<th></th>
<th>open</th>
<th>read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Instances</strong></td>
<td>20,807</td>
<td>20,101</td>
</tr>
<tr>
<td><strong>Number of distinct paths</strong></td>
<td>559</td>
<td>636</td>
</tr>
<tr>
<td><strong>Number of interrupted paths</strong></td>
<td>526</td>
<td>591</td>
</tr>
<tr>
<td><strong>Number of Non-interrupted paths</strong></td>
<td>33</td>
<td>45</td>
</tr>
<tr>
<td><strong>Freq. of Most frequent path</strong></td>
<td>21.37%</td>
<td>30.13%</td>
</tr>
<tr>
<td><strong>Proportion of top 7 paths</strong></td>
<td>94.18%</td>
<td>88.84%</td>
</tr>
<tr>
<td><strong>Proportion of top 10 paths</strong></td>
<td>96.70%</td>
<td>92.32%</td>
</tr>
</tbody>
</table>
Path Occurrence Frequency Plots

Figure 3: *open* System Call

Figure 4: *read* System Call
Is the taking of a path dependent on previous paths?

- We sought to test if the paths were independently taken
  - treated the data of *path taken in system call invocation order* as a time series
  - performed auto-correlation test on the data

![Autocorrelation Plot for the read System Call](image)

**Figure 5:** The Autocorrelation Plot for the *read* System Call
Discussion of Results

1. For repeated execution of system calls, there is both temporal and path variability.

2. Though the execution paths in kernel space exhibit diversity, these are functionally equivalent paths from the application’s perspective.

3. The occurrence frequency data show that some paths are more frequently taken than others, allowing us to classify paths as
   - frequently taken paths
   - rare paths

   **Assumption**: The instructions in the rare paths are not reasonably well tested, and will contain residual systematic faults.

4. From the autocorrelation tests, we can safely assume independence in the taking of paths during a system call run.
Claims

- Based on our preliminary results, we make the following claims
  1. The taking of a rare execution path is a random event
     - An application’s input does not determine the taking of a particular rare path
  2. The probability of two or more non-diverse applications taking the same rare and faulty path at the same time is sufficiently low

- We can leverage on this low probability to build a replicated system for fault detection

We propose on a minimum, a 2oo2 architecture to assure safety
Future Work

- Results achieved from the evaluation of non-determinism is encouraging
- We intend to:
  1. Study inherent diversity in a 2-channel hardware/software environment
  2. Assess the fault detection potential of inherent diversity in a 2oo2 system
     - develop a prototypical safety-related application
     - perform safety assessment based on the requirements of IEC 61508
Conclusion

- Diversity in software is a well established approach to ensure dependability
- We suggest inherent diversity
  1. as an alternative approach to generating software replicas
  2. for fault detection in replicated architectures
- On the basis of our preliminary data, we believe that there is sufficient non-determinism in kernel space to manifest as diversity in replicated architectures
- The next step is to investigate the potential of inherent diversity for fault detection, and quantify it
- The inherent diversity approach has a promising potential
  - in enabling use of complex software, like the GNU/Linux operating system, for safety-related systems
Thank You


