Inherent Diversity for Fault Detection in Complex Hardware/Software Systems

Peter Odhiambo Okech
School of Computing & Informatics
University of Nairobi
pokech@uonbi.ac.ke

Abstract—The safety domain has traditionally used selected processors and dedicated operating systems. With life-cycles of processors dramatically shortening and the transition to multicore being almost universal the safety industry needs to address this changes at the foundations of its strategy. This requires that the community takes advantage of and makes use of the current technologies available in the safety domain. One of the ways is the use of off-the-shelf operating systems. To effectively use complex hardware/software in safety-related systems, a demonstration that such a system is sufficient to ensure the safety function is required. We propose in our work the use of inherent diversity as an architectural means to protect such systems against systematic residual faults. We also seek to define a process to verify and quantify the level of safety of systems based on inherent diversity. The inherent diversity approach has a promising potential to play a role in enabling complex software, like the GNU/Linux operating system, for safety-related systems.

I. INTRODUCTION

The design and implementation of software based safety-related systems are typically based on well proven and conservative technologies. While the simple deterministic execution platform based on a constrained micro-controller with an executive or a simple microprocessor with a restrictive real-time operating system is very attractive, the computing environment is rapidly changing. In the hardware domain, processors featuring performance enhancing features (e.g. caches, pipelines, predictive execution) and multicore technologies are now becoming pervasive in all classes of computing from servers to desktops and even embedded systems.

It is therefore mandatory that safety community address emerging modern processor architectures rather than continuing the use of old technologies like the uni-processor system. For example in the automotive domain, the need for saving in terms of spaces and cost by reducing the number electronic control units (ECU), and the high computational power of advanced safety functionality will result in the use of a few but more powerful ECUs [1]. It is likely that these ECUs will be multicore processors. The managing of the resources of complex contemporary hardware and mediating the interaction of the safety-related application with the hardware, requires a fully fledged operating system such as GNU/Linux.

Modern operating systems are complex software systems. Due to their complexity, the use of off-the-shelf operating systems are perceived to be unacceptable for safety-related systems. One of the reasons advanced is that the residual fault density in a software system is related to its complexity and size as measured by the number of lines of code [2] and would be too high for systems whose failure is potentially catastrophic. Another reason put forward against the use of complex operating systems is that it is infeasible to verify such systems hence there arises difficulties in establishing their integrity.

The technological environment presents us with a challenge of embracing the trends even for systems that perform safety functions. The main question that begs for an answer is this: Can we utilize the complexity of the execution platform to protect safety-related systems against the residual systematic faults in the platform?

Our work is based on the premise that rather than fight complexity, we can exploit it to achieve the goal of ensuring that residual faults in the software platform do not lead to failure of these systems. A well know and effective method of protection against systematic fault in software is the use of diversity [3]. We propose to take advantage of the inherent non-deterministic execution in complex execution platforms to generate diversity at runtime in a replicated homogeneous architecture. Our discussion is presented in the context of safety-related systems with a focus on the generic safety standard IEC 61508 [4] and GNU/Linux. The work we are engaged in specifically targets to:

- Investigate if there is sufficient level of non-determinism in Linux kernel space during the execution of a simple application and if this non-determinism can manifest in diversity for replicated architectures.
- Define a process that can be used extract information required to demonstrate that safety systems based on inherent diversity of complex hardware/software systems are sufficient to ensure system safety.
- Quantify the potential level of coverage of inherent diversity of a 2oo2 system.

The target scenario considered is this: There is a simple safety application compliant to IEC 61508 or one of its derivative standards executing in a complex hardware/software platform. The assumption is that the application is correct and fault free. On the other hand, the complex operating system contains undetected residual faults. The application request for operating system services through the system call interface by invoking a system call. Faults can be detected if in a 2oo2...
Experiment 2

<table>
<thead>
<tr>
<th>Path Characteristics of Sample System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Call</td>
</tr>
<tr>
<td>Execution Instances</td>
</tr>
<tr>
<td>Number of distinct paths</td>
</tr>
<tr>
<td>Number of interrupted paths</td>
</tr>
<tr>
<td>Number of Non-interrupted paths</td>
</tr>
<tr>
<td>Freq. of Most frequent path</td>
</tr>
<tr>
<td>Proportion of top 7 paths</td>
</tr>
<tr>
<td>Proportion of top 10 paths</td>
</tr>
</tbody>
</table>

In engineering, the main approaches to dealing with failures are (1) redundancy to gain fault-tolerance and (2) fail-safety to minimize the impact of failure. The predominant faults in software systems can be attributed to systematic faults, arising from either specification, design or implementation. It is posited that redundancy in a software based system means replicating the faults in each of the copies. Diversity in design has been suggested as a solution [5], [6]. Though there are debates about its effectiveness, the use of diversity for protection against systematic faults is accepted. However, the high cost of generating diversity at design time has made it a preserve of only very high integrity systems. To deal with this, automated diversity for example through the use of a compiler has been suggested, and this has been successfully used to protect systems against hardware faults [7] and against attacks in the security domain [8]. Another possibility of generating diversity is to take advantage of runtime property of replicated systems. This is what we refer to as inherent diversity, which provides a promising means to protect systems against residual faults that are expected in complex systems [9].

### B. Complexity and Non-determinism

The computing execution platform operate through the interaction of many parts, both hardware and operating systems components. The interactions leads to coupling of the elements of the systems, resulting into global properties that cannot be attributed to any of the components in isolation. This leads to some level of unpredictability in the internal behavior of an application executing in such complex platform with respect to the paths and timing of system services. In the absence of errors, this unpredictability does not impact the correctness of the system function.

Currently, we focus on the non-determinism in the execution path of an application in kernel space. We seek to (1) determine if there is indeed non-determinism, and (2) if it is sufficient to manifest as diversity in a general N-out-of-M (NooM) architecture with a voting mechanism, that is we have replicas of the same application running in M different hardware/software channels, and the correctness of the function is achieved when N of these channels agree. Our investigation will be restricted to a 2oo2 system, which we believe is strong enough to ensure a safety function. In such a configuration, if the two channels disagree, then the system is moved to a safe state.

### III. Evaluation of Non-determinism

#### A. The experiments

To determine if there is path non-determinism in the Linux kernel, we performed two separate experiments. The experiments involved setting up kernel path tracing environment using the tool Ftrace [10]. In the first experiment, we traced the execution of the `open` system call for a large number of invocations from a trivial c-program. For the second experiment, we created an application that reads a value stored in a binary file, increments the value and writes the new value to the file. This set of statements were executed in 20,100 iterations, and the kernel path traces recorded. The set of system calls `{read, write, lseek, fsync, nanosleep}` were repeatedly invoked by the test program. Table I shows the characteristics of the paths taken during the execution in of the system calls `open` from the first experiment and `read` from the second experiment. From the 20,807 execution instances of the `open` system call, there were 559 unique paths. The most frequent path was taken by 21.37% of the execution instances. Similarly from 20,101 execution instances of the `read` system call, there were 636 unique paths. The most frequent path was taken by 6,057 or 30.13% of the execution instances.

From the data gathered in the experiments, we generated a plot of the frequency of the paths taken by a system call during execution instances. Figures 2 and 3 shows the plots from experiment 1 and 2 respectively. Note that in both cases, the snapshot of the frequency plot does not show all the paths with a occurrence frequency of 1.
possibility that the rare paths are not reasonably well tested, paths in a complex system. This situation leads to a very high the paths into two groups: frequently taken paths and rare paths. We contend that a test campaign on complex systems by discussions those paths that are interrupted, we can classify every time a system call is invoked.

From the frequency data, and ignoring for the purpose of path variability of a system call from repeated execution of a program. For most non-trivial system calls (an example of a simple system call is lseek from our second experiment), there exists several possible execution paths through the kernel space. From the point of view of an invoking application, these are functionally equivalent paths, but internally and transparent to the application, they exhibit path diversity. We can attribute the path diversity to 1) preemption of the system call and 2) the internal state of the operating system, which is different every time a system call is invoked.

The results obtained show that there is both temporal and path variability of a system call from repeated execution of a program. For most non-trivial system calls (an example of a simple system call is lseek from our second experiment), there exists several possible execution paths through the kernel space. From the point of view of an invoking application, these are functionally equivalent paths, but internally and transparent to the application, they exhibit path diversity. We can attribute the path diversity to 1) preemption of the system call and 2) the internal state of the operating system, which is different every time a system call is invoked.

From the frequency data, and ignoring for the purpose of discussions those paths that are interrupted, we can classify the paths into two groups: frequently taken paths and rare paths. We contend that a test campaign on complex systems within a reasonable budget will only exhaustively test the most frequently encountered paths. We believe that it is not feasible using current techniques to write test cases to test all possible paths in a complex system. This situation leads to a very high possibility that the rare paths are not reasonably well tested, and hence will contain residual systematic faults.

Suppose there is a latent fault in one of the rare paths of the complex operating system. What is the probability that during an execution of a non-diverse safety application running on a 2-channel complex hardware/software system, the two systems taking the same faulty path? We believe that this probability is sufficiently low. Essentially the rare path thus exhibits properties of a random fault and thus can be mitigated with traditional architectural methods. We can leverage on this low probability to build a replicated systems based on a 2oo2 logic for fault detection to provide safety for even highly critical systems. What is needed is a quantification of the level of safety that can be attained through the inherent diversity mechanism.

An important question that we sought to answer is if there is correlation between the paths taken during an execution run. To answer this question, we performed tests on the data of the paths in system call order, i.e. the paths sorted in the order of their invocation, using the autocorrelation function. The plot of autocorrelation coefficient against the lag in the range 1..5025 for the read system call of experiment 2 is shown in Figure 4. Though we cannot claim that the system call paths are truly independent, due to the low magnitudes of the autocorrelation coefficients it is still possible to safely assume independence so as to investigate the inherent diversity technique.

B. Variability in the execution path

An important question that we sought to answer is if there is correlation between the paths taken during an execution run. To answer this question, we performed tests on the data of the paths in system call order, i.e. the paths sorted in the order of their invocation, using the autocorrelation function. The plot of autocorrelation coefficient against the lag in the range 1..5025 for the read system call of experiment 2 is shown in Figure 4. Though we cannot claim that the system call paths are truly independent, due to the low magnitudes of the autocorrelation coefficients it is still possible to safely assume independence so as to investigate the inherent diversity technique.

Fig. 2. Frequency Plot of the open System Call Paths from Experiment 1

Fig. 3. Frequency Plot of the read System Call Paths from Experiment 2

IV. CURRENT STATUS AND FUTURE WORK

The results achieved from the repeated execution of simple programs is encouraging. The experiments were performed in more or less idle systems, with noticeable levels of path non-determinism for most of the systems calls investigated. Our next step is to determine if this non-determinism will manifest as diversity in a replicated architecture.

To achieve this, we are currently working on a configuration of two loosely coupled machines concurrently executing the same program and by extension invoking the same sequence of systems calls. We intend to use this setup to study inherent diversity in a 2-channel hardware/software environment. The focus of our investigation will be 1) the assumption of independence of the runtime execution of replicated software in complex execution platform, and 2) the percentage of potential failures that can be attributes to common cause faults.

We are also in the process of developing a prototypical safety-related application using safety engineering principles, which we will deploy in the 2oo2 configuration. The availability of a realistic safety-related application will inform our future work. The application will be used as a case study to achieve our main objective, the assessment of the fault
detection potential of inherent diversity in a 2oo2 system in a safety context.

The aim is to use the case study to demonstrate that the total system based on inherent diversity of a 2oo2 architecture with a safety application running on top of a complex hardware/software system is sufficient to ensure system safety. This can be achieved by defining a reasonable reproducible and sufficiently assured process to quantify the safety level achievable by the use of inherent diversity. We will document the validation and verification process of the system based on inherent diversity in the context of the IEC 61508 safety lifecycle with a view to make a contribution to the support required for functional safety validation and certification effort of systems based on complex hardware/software systems.

V. Conclusion

Diversity in the software stack is a well established approach to ensuring the dependability of safety-related systems. We have suggested the concept of inherent runtime diversity for residual fault detection as a method for supporting qualification of complex hardware/software systems in the safety domain. In addition, it is an alternative approach to generating replicas in response to the limitations of traditional means of achieving diversity. Our preliminary work suggests that there is a sufficient level of non-determinism in the execution of simple applications in kernel space that would manifest as diversity in replicated architectures. We intend to follow this up by developing a strategy for the verification and assurance of safety-related systems based on the inherent diversity concept. This would result increased acceptance of complex software, like the GNU/Linux operating system, for safety-related systems.

Acknowledgment

The author would like to thank Nicholas McGuire from OpenTech EDV Research GmbH Austria for the invaluable insights he provided, Prof. Dr. Christof Fetzer of TU-Dresden and Prof. William Okelo-Odongo of University of Nairobi for their help and guidance. The author gratefully acknowledges support from the Mundus ACP II PhD Mobility Scholarship.

References


