



Exhaustif: A fault injection tool for distributed heterogeneous embedded systems

Antonio da Silva José-F Martínez Lourdes López

Luís Redondo

Dept. of Telematic Engineering and Architectures
Universidad Politécnica de Madrid
Crta Valencia, Km 7, 28031 Madrid, Spain
{adasilva, jfmartin, llopez}@diatel.upm.es

Métodos y Tecnología
Po. Castellana 182,
28013 Madrid, Spain
lredondo@mtp.es

Abstract

This paper presents a new fault injection tool called Exhaustif (Exhaustive Workbench for Systems Reliability). Exhaustif is a SWIFI fault injection tool for fault tolerance verification and validation of embedded software in distributed heterogeneous systems. Mainly Exhaustif consists in two parts: EEM and FIK. Exhaustif Executive Manager (EEM) is a GUI Java application to define the faults injection campaign that uses a SQL database to save the test results obtained from System under Test (SUT) in order to perform post injection data analysis. A Fault Injector Kernel (FIK) for an EADS-Astrium SPARC ERC32-based MCM processor board has been developed. FIK is under EEM command to perform faults injections in applications running under RTEMS operating system using pure SWIFI techniques. Exhaustif performs register and memory corruptions with temporal triggers and introduced an optimized routine interception mechanism to perform argument and return value corruption with minimal time overhead. EADS-Astrium (France) is a world leader in the design and manufacture of satellite systems; its activities cover complete civil and military telecommunications and Earth observation systems. This work is led by Métodos y Tecnología (MTP: www.mtp.es) in collaboration with Universidad Politécnica de Madrid –(DIATEL) and co-financed by the Ministry of Economy and Technologic Innovation at the Autonomous Region of Madrid (Spain).

Keywords: *Fault Tolerance, SWIFI, embedded systems, heterogeneous systems*



1 Introduction

Study of failures and errors is an important part in the evaluation of system reliability. To understand the potential failures, there have been developed experimental techniques that can be applied both to the hardware and to the software. These techniques not only are suitable during the phase of system analysis and design, but also during their prototyping and manufacturing phases, in other words during the whole of system life cycle.

In the current society, every time, computer systems that control any process receive more importance, from a simple domestic appliance up to auto-motion control system of bigger bank systems or those of telecommunications. Sometimes many human lives depend on the correct functioning of these applications. This is the case of traffic control systems, equipments of vital support or nuclear power centrals. Obviously, these real time applications, not only must guarantee that the awaited result is of the correct value, but it is necessary to deliver it in the specified term.

There exist some examples of “well-known” systems failures with important negative repercussions. For example:

- The blackout of America On line (AOL) that affected six million users.
- The destruction of Ariane5 due to software failures, which costs 250 trillions of dollars to the European Spatial Agency (ESA) [1].

The reliability requirements of these kind of systems cannot be reached exclusively by techniques or technologies of fault prevention (design, insurance of the quality, shielding, etc.), since the hypothesis of being able to construct an absolutely fault free system is not realistic [2]. Therefore, it is necessary to construct systems that are capable of giving the waited enclosed service before the presence of faults. It is possible to do validation of reliability attributes in a system in two ways: theoretical or experimental. The theoretical evaluation is realized evaluating the system model, while the experimental validation is realized by means of the real observation of the system in normal functioning while there are injected failures of deliberate form.

The injection of failures can discover errors that normal procedures cannot. First, it tests the mechanisms of exception and treatment of failures that in normal circumstances are not sufficiently proven and, helps to evaluate the risk, verifying how much defective can be the system behaviour in presence of errors. All of the injection failures methods are based on concrete hardware/software characteristics associated to systems which are applied, then, to realize generalizations is a very complicated task.

The *ExhaustiF* tool (*Exhaustive Workbench for Systems Reliability*) here presented serves precisely to exercise the failure tolerance mechanisms introduced in a system.

1.1 SWIFI TOOLS

Software Implemented Fault Injection technique (SWIFI) is a very attractive since it does not need specific hardware to realize the fault injection. It can be used to prove the failure tolerance mechanisms of applications, operating systems and, in general commercial off-the-shelf software components (COTS) of which the source code is not available. Use of COTS and code reuse [3] is increasingly frequent which complicates the evaluation of exception mechanisms. If



the target of test is an application, the injector is inserted in the same application or in an intermediate layer between the own application and the operating system.

On the other hand, if the target is the operating system, the injector must be absorbed in the same operating system, and when this turns out to be very complicated it is necessary to add an intermediate layer (wrapper) between the system and the hardware.

Some advantages of SWIFI techniques for fault injection can be out-standing:

- The temporary intrusion can diminish to necessary used time in the exception handling and can be rejected, if temporary restrictions of system under test are not strict.
- It is possible to carry out many experiments in a simple and controlled form.
- The experiments are carried out on real hardware.
- No special equipment is necessary and it is easily adaptable to tested target system.

The main characteristic of fault injection software is that it is capable of injecting failures into any functional addressing unit by means of software, such as memory, registers, and peripherals. The goal of the fault injection software is to reproduce, in the logical scope, the errors that are reproduced after failures in the hardware. A good characterization of failure model should be allowed that this one was as versatile as possible, allowing a major number of combinations among the location, trigger conditions, kind of fault and duration, so that the coverage was maximum.

The failure model characterization is carried out limiting where, when and during how long time has to be injected the fault and which kind of type has to be it.

When failure injection is carried out with temporal triggers and corrupting parameters in routine calls, Exhaustif introduces a temporary overcharge equivalent to the execution of a few machine instructions, negligible for the majority of applications.

The remainder of the paper is organized as follows: more important SWIFI tools are introduced and analyzed in section 2. Section 3 explains the ExhaustiF tool, its architecture and components, its design decisions and implementation. In section 4 concludes this paper with an outlook on future research activities.

2 Related Work

Several tools have been created for fault injection by using pure SWIFI paradigms or in conjunction with hardware debugging resources present in some processors, being the generalization for any kind of system very complex. Some tools could be aforementioned: XCEPTION, GOOFI, MAFALDA or FAUMachine.

- Xception [4] is fault injection tool by means of software and it has a Scan Chain interface (SCIFI) for ERC32. When it works as a pure SWIFI it tries to use the debugging and monitoring characteristics, which are presented in the majority of the processors to inject more realistic faults. Besides, it monitors the fault activation and its impact in the system, showing its detailed behaviour. Xception provides a set of fault injection, including spatial, temporary faults and faults related to the information manipulation in memory. It is based on the utilization of debugging mechanisms implemented on the processor across exceptions in order to realize the injection. First it allows executing the application until trigger time, then a track mode is available, it is executed step by step until the instruction to inject. Once that



instruction is reached, it is decoded and decided that parameters of the same one to modify depending on the type of failure that is tried to inject. By using SWIFI or SCIFI Xception diverse CPUs are supported such as: PowerPc750, Intel x86 and Sparc ERC32 on diverse operating systems: LynxOS, VxWorks, Linux, and Win32. Also it has been used to verify RTEMS's robustness [8].

- Goofi [5] (Generic Object-Oriented Fault Injection) is designed to be able to adapt to several systems under test and to different fault injection technologies. It allows realizing fault injection by means of software (SWIFI) and Scan Chain (SCIFI). Besides it possesses BDM (Background Debug Mode) and Nexus interfaces. Goofi supports Thor Rard Hard microprocessor of SAAB Ericsson Space, the MC68340 and HC12 microprocessors, and the MPC565 microcontroller, both of Motorola.
- Mafalda [6] (Microkernel Assessment by Fault injection Analysis and Design Aid) is orientated to the microkernel validation (operating system kernel) and allows to realize two types of injections. The corruption of input parameter when a microkernel primitive is invoked and the failure injection in memory segments, of code and information. In case of the injection in system call parameters, calls are intercepted by means of a library (wrapper) of calls modified for the injection, or by means of a breakpoint in the microkernel input, doing that executes, before call, a driver in charge of corrupting the aforementioned parameters. In case of the memory injection, code or information, there are defined permanent or transitory failures triggered by spatial or temporary events, where bits chosen carefully are reversed, by means of with this technique is feasible to emulate hardware faults in the memory, or emulate software failures by modifying input parameters in the kernel. In case of transitory failures, after the injection a track mode is available, then execution is done step by step and memory is restored.
- FAU Machine [7] is a virtual machine that emulates PC/AT hardware and its peripherals. It has as advantage on other hardware simulation systems which allow simulating the malfunction of someone of their components, for example: memory. Virtual machine use allows simulating permanent failures, which would not possible by means of SWIFI techniques. Use of Simulation environments is indispensable in order to realize injection of hardware permanent errors.

Hardware debugging mechanisms use is very interesting and indispensable for spatial trigger specification, but it presents problems in distributed heterogeneous environments. When a system has detected a trigger, it goes to a frozen state allowing the analysis of its state by means of interfaces such as Boundary-Scan. In this case it is necessary to warn other systems of this event; in order to stop, also, their activity. Exhaustif is a tool designed for being used in distributed heterogeneous environments. It makes difficult the use of particular debugging systems, being SWIFI the easiest option in order to adapt to different environments.

Wrappers [9] use is only possible if application source code is available and the insertion of software break points introduces a valuable temporary overload. Exhaustif modifies the routine entry point by inserting a jump to a block of code defined specifically for the failure that is wanted to insert, coming back immediately to the code of the routine avoiding the exceptions mechanism.



3 Exhaustif tool

Exhaustif is designed to analyze dynamically systems in which it is necessary to validate if reliability requirements are satisfied. Due to flexible validation of target system it is possible to test fault tolerance mechanisms and appropriate management of faults can be verified according to its specification

The hardware/software Exhaustif architecture has been designed to reduce to the maximum the impact of any change (incorporation, modification or erased of any module). This designed tool consists of two principal elements:

- EEM (Exhaustif Executive Manager)
- FIK (Fault Injection Kernel)

EEM is in charge of management and control of experiments execution. It offers a graphical user interface in order to specify and program the injections of fault in an understandable and simple manner, as well as result visualizations and information about realized executions.

FIK is a lightweight software component which resides in the System Under Test (SUT) and it has the function of injecting faults. For describing the Exhaustif's Fault Injection Kernel in this paper we will use a FIK version which gives support to a main-board based in ERC32 facilitated by EADS-ASTRIUM [10] and executes applications under RTEMS [11].

3.1 Architectural design

In EEM design we have taken in account its capacity for supporting simultaneously several SUTs. In this way would be possible to inject faults and/or errors into several systems under test, which in turn form a part of a bigger system, verifying in so far as injected failures in a subsystem affect into other one. In detail, Exhaustif is composed of the following components (see Fig. 1):

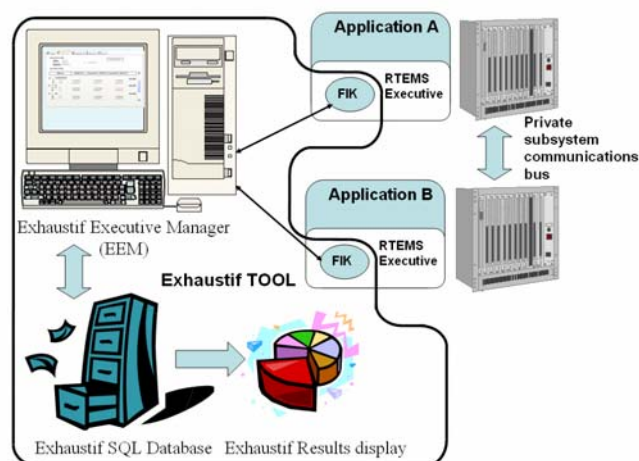


Fig. 1. ExhaustiF architecture and components

- ExhaustiF Executive Manager - EEM is in charge of management and control of experiments execution. It allows the user to configure the test cases and select failures which will be injected. The EEM arranges the sequence of failures to be injected inside the SUT. Finally, it



gathers execution information and obtains results for visualization. In the EEM design and development portability for different execution platforms both hardware and operating systems has been taking in account, thus a java implementation was done.

- Fault Injector Kernel - FIK is a lightweight software component. It is executed in the platform of system under test and is in charge of injecting failures. Since the FIK is a component of very small size, practically not intruder (see 3.3 section) allows to provoke faults in the memory, in CPU registers, of floating comma unit and allows intercepting functions calls in such a way that input arguments and returned function value can be modified.

Besides the components above-mentioned, Exhaustif has others very important as follows:

- A dedicated communication line connecting the Agent EEM and the Injector FIK. In the Exhaustif design there has been done a communication system abstraction between EEM and FIK in order to permits a complete flexibility in the communications depending on the current SUT capacities. Then, Exhaustif is sufficiently flexible to admit several communication interfaces such as RS-232-C, USB, Ethernet, etc.
- A Standard SQL database that contains information collected in the execution of system under test.
- Tools for statistical analysis and information visualization of stored information in the ExhaustiF database.

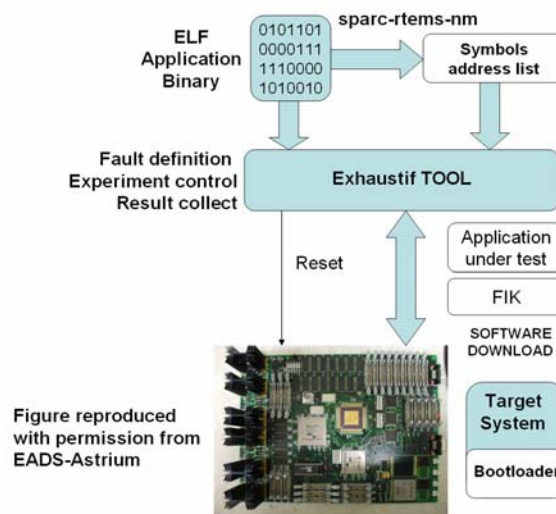


Fig. 2. Operation mode

After system start, FIK and application under test are loaded by using the starting monitor services included into the board (see Fig. 2). Once started, the FIK is waiting for an EEM message with failures to insert. When incubation time has passed, EEM requests to FIK the specific information that it wants to store in the database.

It is possible to execute an experiment without failure injection (Golden Run), that only request information messages are sent to the FIK, which will be stored in the database with intention of comparing them, later, with the obtained ones in the executions with errors and then to observe the produced changes.

3.2 Exhaustif Executive Manager

The definition of the testing workbench in Exhaustif is realized across projects. A project will be composed of one or more experiments (see Fig. 3). Every experiment is composed of injections and these ones in turn by failures, for every failure are specified the temporary trigger and the incubation time after which it is possible to request the injection results.

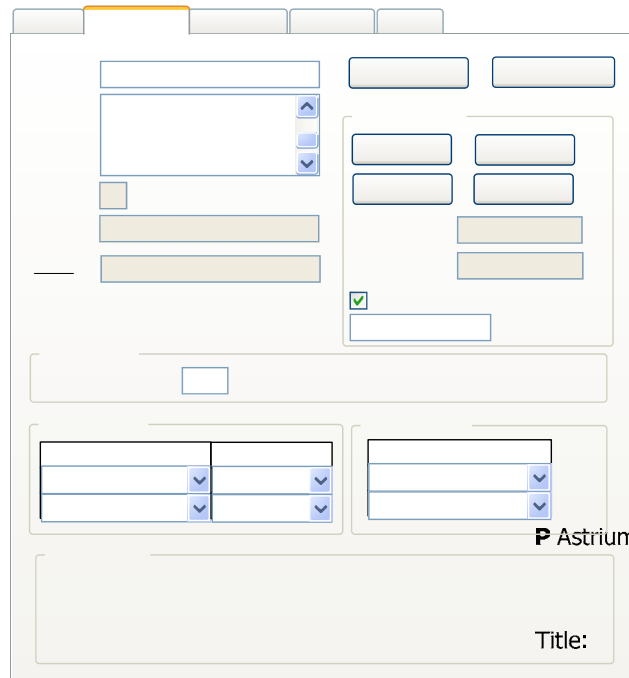
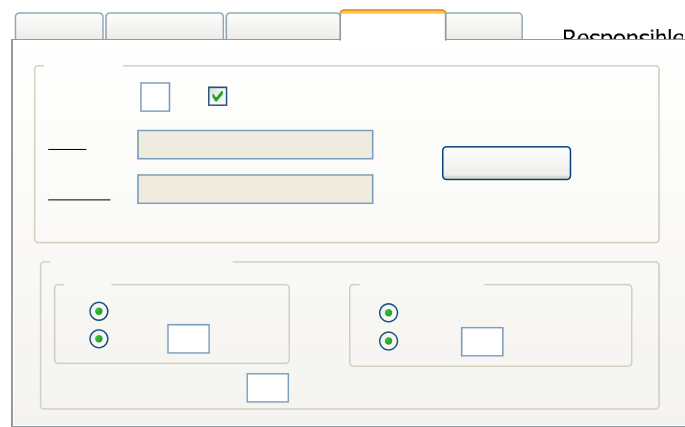


Fig. 3. Experiment definition

P Astrium
 E Experiment 2 E Experiment 3 I
 Title: Experiment with Memory Fault
 Description: Brief Project Description

For every injection the injection time is defined (see Fig. 4). It could be of type *now* (in the same time in which it sends the injection message) or a *timeout* from the beginning of the application and in the incubation time, which will be waited before requesting the content of test points, specified in the experiment.



Responsible: Luís Miguel García Olmos
 Astrium
 Number: 2
 Parameters
 Sampling Frequency: 5000 ms
 Memory Monitoring

Memory Address	Block Size (byte)
0xA6BE0990	8
0x45A6FF99	16

Fig. 4. Time trigger definition

In a graphic manner, the sequence diagram in Fig. 5 shows the interchanged messages between EEM and FIK for an injection *now* and its relationship with the incubation time.

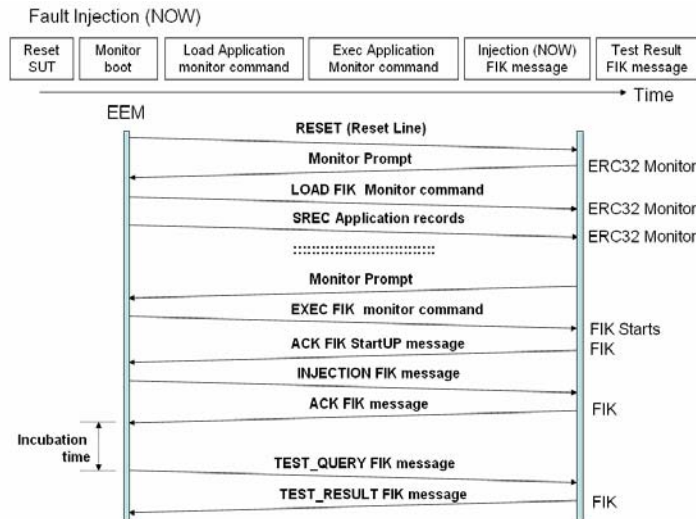


Fig. 5. EEM/FIK messages interchange

Exhaustif allows specifying corruptions of memory and processor registers by applying different modification patterns, such as changes in the state of a bit (BitFlip), use of a mask (BitMask) or copy of a new value (see Fig. 6). These modifications can be realized instantaneously in the moment that they are ordered or defining a timeout from the beginning of the application. All of these modifications are instantaneous and not permanent. On the other hand, Exhaustif allows intercepting functions calls (see Fig. 7), by altering the content of the input parameters and/or the returned value with a minimal overloading of execution. See section 3.3.



Fig. 6. Memory and variable corruption specification

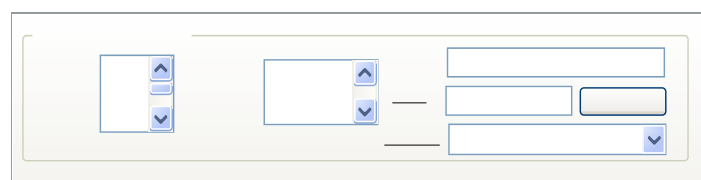


Fig. 7. Routine and call interception

Finally, after injection of all specified failures, the injections that have differences with regard to execution without errors are visualized. Additionally it is possible a temporary visualization of test points contents obtained during the execution of all experiment injections (see Fig. 8).

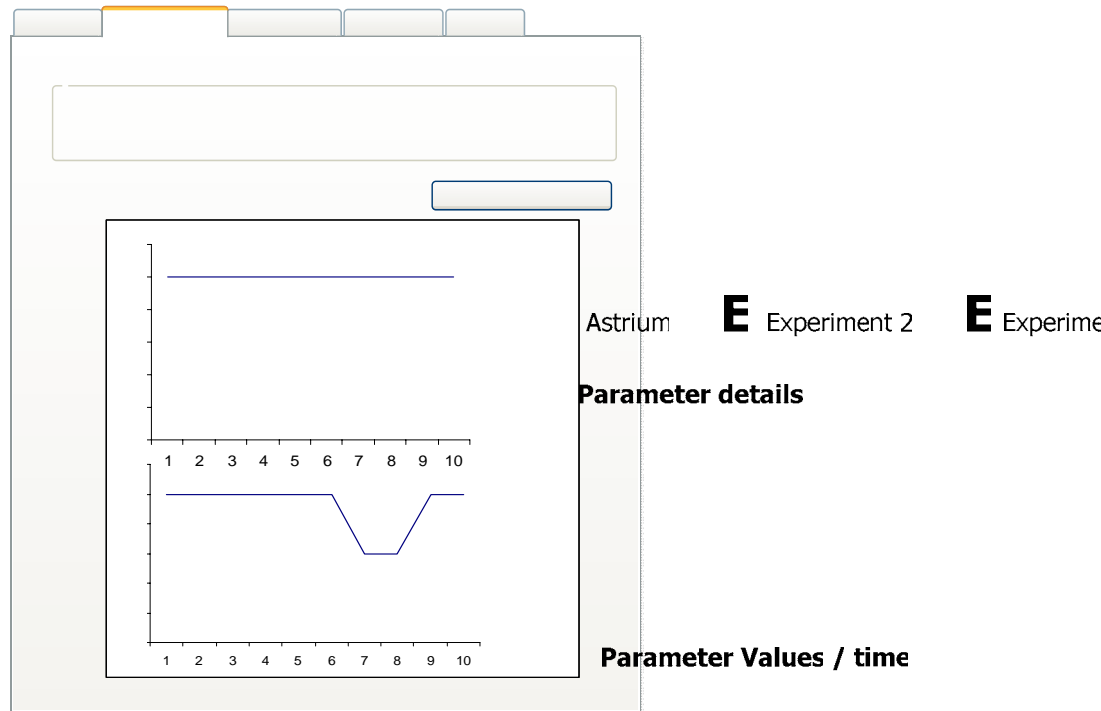


Fig. 8. Results timeline

3.3 Fault Injector Kernel

FIK is the software component in charged of injecting the specified faults in the EEM. Design conditions impose that it must be practically not intruder and to occupy a maximum of 128 Kbytes. On the other hand, the FIK is totally application independent and works under EEM orders with the aim of carrying out programmed injections in the different experiments.

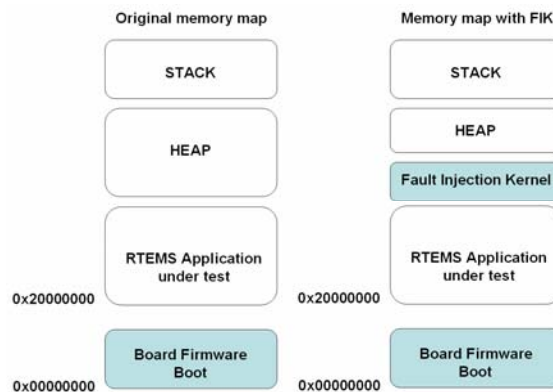


Fig. 9. FIK/APP memory map

As it was mentioned in previous paragraphs, FIK is a lightweight software component that in communication with the EEM and accomplishes the programmed injection of failures. In the concrete case of ERC32 architecture and under RTEMS operating system, the memory map of system under test would be the showed in the Fig. 9.



- The application is loaded from 0x20000000 memory address. The FIK is loaded after application and it uses the same communication line that starting monitor in order to communicate with the EEM.
- FIK occupies part of memory initially dedicated to "heap", and therefore it modifies the configuration of C standard library in order that dynamical memory services are adapted to the new configuration.
- To give service to failure injection with temporal trigger specified in the injection message, the FIK captures the service to real time interruption programmed by RTEMS (see Fig. 10).

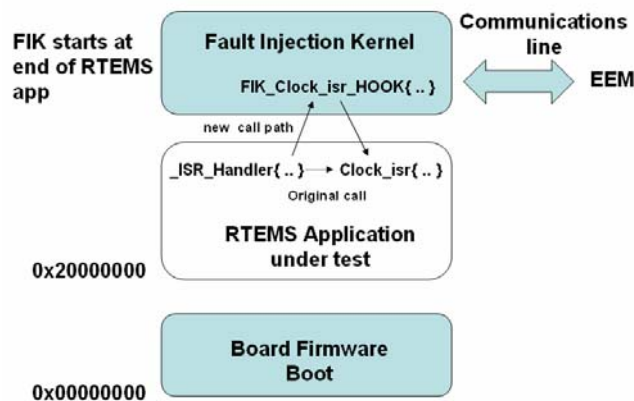


Fig. 10. Timer ISR Hook

Election of a representative set of failures is a complex task. It is possible to achieve memory corruptions and not to provoke any failure, this can be because these are not being used by the application or, because the corrupt value is not used and it is rewritten. Exhaustif in addition to memory and register corruptions, also, allows the interception of routine calls and the corruption of their input arguments, as well as the alteration of return value. By means of corruption of input parameters it is possible a better location of injected error and API checking robustness.

In classic schemes a software break point (TRAP) is inserted at the beginning of the routine address and, in the exception handling an argument modification is carried out and immediately after the execution is restarted (see Fig. 11).

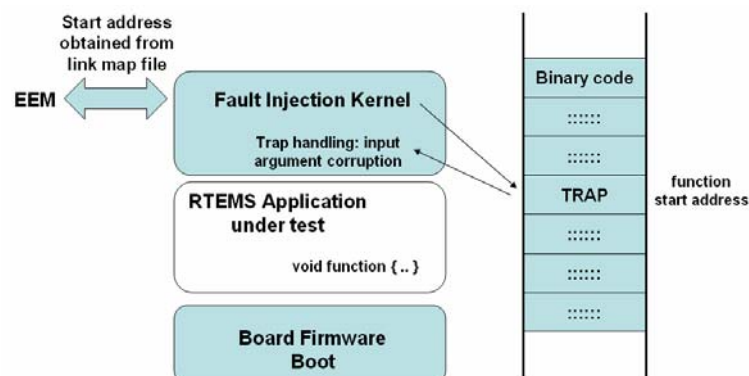


Fig. 11. Traditional approach

This scheme is very comfortable but especially costly, in temporary terms, in the Sparc architecture where the prototype has been developed and implemented. The Exhaustif FIK

achieves, on-demand, a mutation of the original code by inserting an unconditional jump code to a block of code when the EEM requests it. The block of code is in charge of modifying arguments and comes back again to the code of the original routine (see Fig. 12).

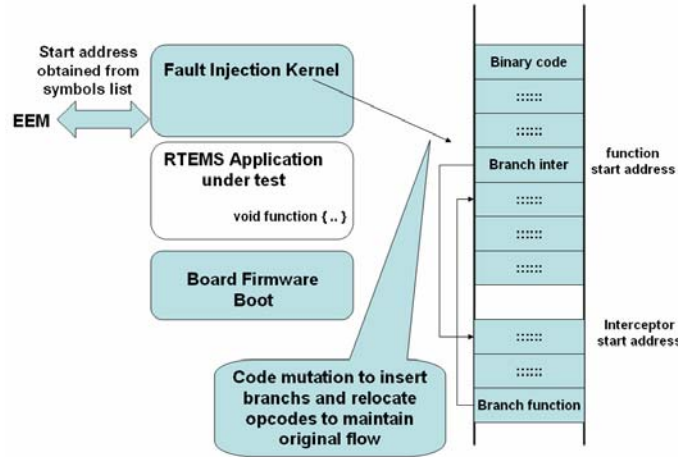


Fig. 12. Fast routine interception

Simulations carried out with TSIM, ERC32/Leon simulator [12], in which a mask is applied on the second argument of a function, an overload of only 17 cycles of clock is provoked. Using TRAPS are necessary at least 25 cycles of clock to jump to the beginning of the code that provokes the corruption. Furthermore, in case of Sparc, inside this routine would be necessary to add the register windows management (see Fig. 13).

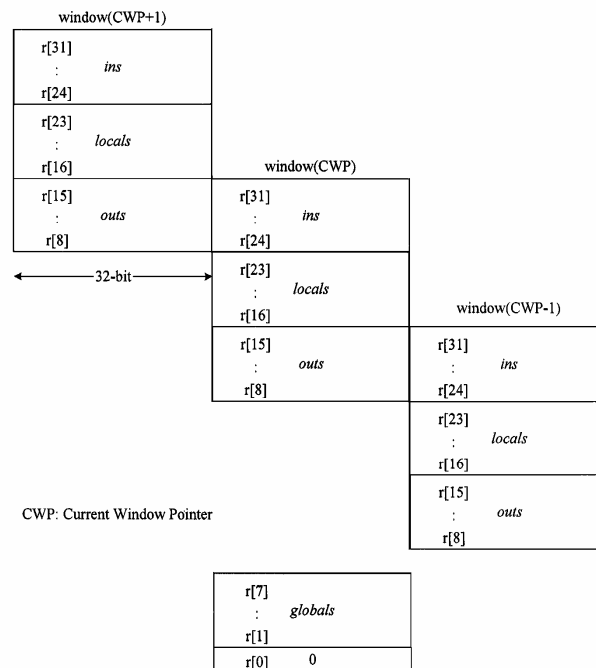


Fig. 13. Sparc Register Window

Sparc uses intensively the register block as mechanism of input parameters to functions, storage of return address and function returned value. Then a sliding window is uses in order to



modify the set of registers, seen from software in every moment. At one stage the programmer has to his disposition 32 registers:

- 8 global registers.
- 8 input registers (ins).
- 8 local registers (local)
- 8 output records (outs)

By means of couple of instructions SAVE/RESTORE it is possible to move the current window forward or backward. On having advanced, the outs registers of the previous window is converted into "ins" registers in the current window.

According to Sparc Application Binary Interface (ABI) [13] the routine calls in C have the following agreement: the function arguments are put in the output registers [O0...O5], in O7 is put the address of function call instruction. If function needs a new register window of records, it will execute an instruction save that provokes the window movement. In case of final functions which do not invoke to any routine, it is not necessary to move the register window.

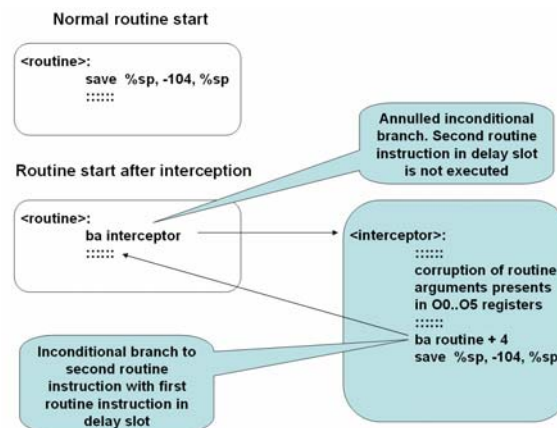


Fig. 14. Interceptor code

In this way actions carried out by FIK to achieve an interception will be the following ones (See Fig. 14):

- To replace the first instruction of the function with an unconditional jump at the beginning of interceptor code.
- To corrupt the register content which contains the parameter value that wants to be changed and jump again to the function code.

4 Conclusions and future work

The capacity to inject faults by means of software constitutes a tool of high interest for verification of fault tolerance mechanisms which are foreseen in the construction of critical systems, as well as to predict the consequences of the bad systems behaviour due to errors not detected in functional tests. Furthermore, this technology is applicable in third party software component validation, without having access to component source code. Use of Exhaustif contributes to the task of constructing quality software.



The design has been done in order that the migration to other environments can be easily achieved. Likewise, some scenarios are being evaluated in order to explore Exhaustif capability to control simultaneously several SUTs and to realize tests in which injection and results gathered are not carried out on the same system under test.

An automatic way of experiment generation is being designed and implemented for Exhaustif. This is defined depending on statistical models of memory and register corruptions and provoked by SEUs and EMC. In this way failures, where they are provoked and their duration time would come determined by these models and would be automatically selected by the tool.

Other future works consist in implements the FIK to other hardware platforms such as i386, Motorola ColdFire and/or operating systems such as uCLinux.

References

1. Voas, J.: Software Fault-Injection: Growing 'safer' Systems, In Proceedings. of IEEE Aerospace Conference, Snowmass, February (1997)
2. Carreira, J and Silva, J.G.: ¿Why do some (weird) people Inject Faults?, Software Engineering Notes, Vol 23 no 1, pp. 42, January (1998)
3. Gacek, C., de Lemos R.: Architectural description of dependable systems, In: Structure for Dependability: Computer-Bases Systems from an Interdisciplinary Perspective. Lecture Notes in Computer Science 4527. Springer-Verlang, pp. 127-142, (2006)
4. Carreira, J., Madeira, H., and Silva, J.G.: Xception: A technique for Experimental Evaluation of Dependability in Modern Computers, IEEE Transactions On Software Engineering, Vol. 24, pp 125-135, February (1998)
5. Aidemark, J., Vinter, J., Folkesson, P., and Karlsson, J.: GOOFI: Generic Object-Oriented Fault Injection Tool, In Proceedings of International Conference on Dependable Systems and Networks (2001).
6. Arlat, J., Crouzet, Y., and Laprie, J.C.: Fault Injection for dependability validation of fault-tolerant computing systems, Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S, Toulouse, France (1989).
7. Höxer H.-J., Sieh V.; Waitz M.: Fast Simulation of Stuck-At and Coupling Memory Faults Using FAUmachine, In Supplement to Proceedings. HASE 2005: International Symposium on High Assurance Systems Engineering (2005)
8. Maia, R., Enriques, L., Barbosa, R., Costa, D., and Madeira, H.: Xception Fault Injection and Robustness testing Framework: a case-study of testing RTEMS, WTF: VI Workshop de Testes e Tolerancia a Falhas (2005)
9. Arlat, J., Fabre, J.C., Rodriguez, M., and Salles, F.: MAFALDA: A series of prototype tools for the assessment of real time COTS microkernel-based systems, In Fault injection techniques and tools for embedded systems reliability evaluation, Boston: Kluwer academic, (2003)
10. EADS-Astrium: www.astrium.eads.net (2006)
11. RTEMS: Real Time Executive for Multiprocessor Systems. www.rtems.org. (2006)
12. TSIM: ERC32/Leon Simulator. www.gaisler.com (2006)
13. Sparc Standards: www.sparc.org (2006)