Robustness study of an embedded operating system for industrial applications[1]

Pardo, J., Campelo, J.C, Serrano, J.J.

Fault Tolerant Systems Group
Polytechnic University of Valencia.
Camino de vera, s/n. 46022 Valencia, Spain.
juaparal@upvnet.upv.es, jcampelo@disca.upv.es

Abstract

Critical industrial applications or fault tolerant applications need for operating systems (OS) which guarantee a correct and safe behaviour in spite of the appearance of errors. In order to validate the behaviour of an operating system, software fault injection techniques can be used. These techniques could be used to corrupt the information of some of the operating system calls to see how the system reacts in front of invalid or corrupted values at the kernel calls. The research work presented in this paper is about development and preliminary results obtained from the experimentation on software fault injection in an embedded system composed by a Real-Time Operating System (RTOS) like MicroC/OS-II and a microcontroller as the Infineon C167. A software fault injection tool has been developed. The methodology proposed treated the operating system as a black-box where the source code was not available. With this objective a layer between the operating system and the application to be executed has been developed. OS error detection coverage has been measured and observations about OS critical data structures to be improved have been commented, in order to improve the final robustness of the OS

1. Introduction

In order to measure the quality of the software some tests are required. Fault tolerance deals with software's ability to hide problems, specifically the effects of the faults [3]. Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness can thus be viewed as an indication on the OS capacity to resist/react to faults induced by the applications running on top of it, or originating from the hardware layer or from device drivers [2].

The research work presented in this paper is about the development of a software fault injector for a commercial-off-the-shelf (COTS) real-time operating system like MicroC/OS-II in an embedded system with an Infineon C167 microcontroller, typical for the automotive industry. The objective is to evaluate the OS error detection coverage, error propagation and error detection latency times in a fast and easy way. It could be very interesting to be able to obtain data about robustness and dependability of the OS, and for the future to be able to compare different COTS RTOS on similar hardware platforms.

2. System Design

The main motivation to use Commercial Off-The-Shelf (COTS) components on a system design is the notorious cost reduction associated to the final product development. The use of COTS components becomes a cost-effective method for rapid prototyping of complex software systems. On the other hand, the use of COTS software components have serious certification problems due to their design process is unknown. COTS software is composed of general purpose components which have poor dependability specifications. Usually, COTS components are like a black-box, the source code is not available and their internal architecture (structure and data flow) is not adequately documented.

The selection of MicroC/OS-II as COTS RTOS came motivated from the perspective that it is a system widely used since several years ago. It is possible to

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

find different applications and systems like industrial robots, motor control, medical instruments, etc., which use this operating system.

The aim of study was to use the black-box approach for the OS study. So the OS source code was not modified trying to avoid as maximum as possible an intrusion in the OS behaviour. With this objective, a layer named as Meta-Kernel, had been developed between the OS and the industrial application to be executed. Through this layer the fault injection was realized in any of the parameters of the system calls to measure the OS robustness. On the other hand, an injection agent developed was in charge of injecting faults and invalid values at the kernel calls in order to monitor the system robustness.

The faultload is the most critical dimension of an OS benchmark and more generally of any dependability benchmark. Two techniques for system call parameter corruption could be used: the 'bit-flip technique' consisting in flipping systematically bits of the target parameters and the 'selective substitution technique' when invalid data values are introduced in the system call parameters. Studies have demonstrated the equivalence of the errors provoked by the two techniques [2].

3. Expected outcomes

Errors obtained after the fault injection could be grouped in two clearly differentiated groups; injected faults which no affected the system and injected faults which affected it. Injected faults producing errors on the system had been classified relating to the detection mechanisms activated after the fault injection.

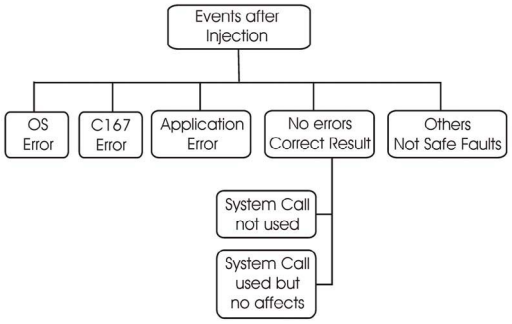

Figure 1. Events after the fault injection

As the figure 1 shows, after the injection of a fault several output results could be expected on the system:

•Operating system error code. The operating system detected something wrong and returned an OS error code at the kernel call execution.

•Infineon C167 error. The microcontroller produced an interrupt which aborted the application execution. An error through the exceptions handler like the stack overflow or underflow, undefined operation code, protection fault, illegal word operand access, etc. was produced.

•Application error. The obtained application result was different from the reference value, i.e. the expected execution value was incorrect.

•Nothing happened. This result meant the injected fault didn't affect the system. The injected fault was hidden by fault tolerance mechanisms of the system. On the other hand, this situation could have been too, when the injection was produced in a system call which finally was not used, which it was not our case.

•And last situation, which was when something happened which produced the system hang, the system didn't react. This situation had been solved through the use of the microcontroller watchdog timer.

4. References

[1] J.Arlat, JC. Fabre, M. Rodríguez, F. Salles. *"Dependability of COTS Microkernel-Based Systems"*. IEEE Transactions on computers, vol. 51, no.2. February 2002.

[2] Pedro Gil , Jean Arlat , Henrique Madeira, Yves Crouzet, Tahar Jarboui, Karama Kanoun, Thomas Marteau, João Durães , Marco Vieira, Daniel Gil, Juan-Carlos Baraza, and Joaquín Gracia.'Fault Representativeness'. Deliverable ETIE2. Dbench European Project. Dependability Benchmarking (IST-2000-25425).

[3] J. Voas, G. McGraw. *"Software Fault Injection: Inoculating programs against errors"*. Edit. Wiley. USA, 1998. ISBN: 0-471-18381-4.

[4] Validated Software Company. "*MicroC/OS-II MISRA C Compliance Matrix"* Application note AN-2004. http://www.validatedsoftware.com. Weston, Florida, October 23, 2002.

([1] This research work has been partially supported by the Spanish CICYT DPI 2003-08320-C02-01 and TIC 2003-08106-C02-01.)