Reliability assessment of FreeRTOS in Embedded Systems

Reliability assessment of FreeRTOS in Embedded Systems

Alberto Bosio[1], Maurizio Rebaudengo[2], Alessandro Savino[2]
[1] Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France
[2] Department of Control and Computer Engineering, Politecnico di Torino, Italy
Email: alberto.bosio@ec-lyon.fr, {maurizio.rebaudengo, alessandro.savino}@polito.it

***Abstract*—This work studies the reliability of a FreeRTOS operating system when affected by Single Event Upset faults. The methodology is based on fault injection to target the most relevant variables and data structures. Results confirm the selectivity in the OS fault tolerance, paving the way to a tailored design of fault-tolerant mechanisms, such as selective hardening of the OS.**

***Index Terms*—Embedded Systems, Real-Time Operating System, Fault Injection, Reliability Assessment, Single Event Upset, Harsh Environment**

I. Introduction

A deep analysis of the system helps achieve System dependability so that, by identifying weaknesses, proper mitigation techniques can be introduced into the system. However, extensive testing phases come at the cost of money and time. For those reasons, during the design phase, the goal is to find an optimal trade-off between mitigation and expenses.

In the literature, all times the whole system is considered, the target is the application-layer [6]. Excluding bare-metal system scenarios, the assumption is that the probability of a fault affecting the middle layer is much lower than the application level due to its shorter execution time. Nevertheless, the lifespan of OS data structures (e.g., process control blocks) is even longer. Moreover, the OS is usually considered a mediator for the data exchange between the hardware and the application, leaving the impact of fault on the application manipulation only. This is only a lucky case.

Authors in [4] proved the above assumption. By performing a fault injection campaign on three different configurations, (i) bare metal, (ii) FreeRTOS, and (iii) Linux OS, they show that the OS significantly impacts the reliability. More particularly, they have shown that the number of observed failures increases with the OS complexity. Therefore, it is mandatory to profoundly investigate the contribution of OS data structure to system reliability.

This work aims to study the reliability of a FreeRTOS [1] when affected by Single Event Upset (SEU) faults. The methodology directly targets all the most relevant variables and data structures of the FreeRTOS operating system to evaluate the system's impact in terms of system integrity, data integrity, and the overall resistance to faults. This kind of comprehensive analysis aims to identify the weak spots in the FreeRTOS so that it can be hardened in different ways, to help designers optimize the error tolerance mechanisms, reducing their impact.

The following sections will present the injection targets (i.e., where faults are injected), the system behaviors classification, and the experimental results.

II. Fault Injection Targets

The fault is selectively injected into OS locations (e.g., variables or data structures). This way, it is possible to analyze the impact of faults on the FreeRTOS. Targets are grouped by their usage, and for each group, we also specify the number of fault locations.

- **FreeRTOS global variables** (*GKVARS* - 10 locations): Global variables used by the kernel to share data among the various kernel functions and to save the state of the system.
- **Task control block (TCB)**: the TCB contains, for each task, all data to guarantee the proper execution of the task, including the context switching. Two sets exist in FreeRTOS: (i) the TCB of running tasks (*CURTCB* - 19 locations) and (ii) the TCB of ready tasks list (*RDYTCB* - 19 locations).
- **Tasks list**: Two lists retain the queue of TCB of task: (i) the ready task list (*RDYLST* - 5 locations) and (ii) the delayed task list (*DLDLST* - 5 locations).
- **Mutex and Queue structure** (*MTXQVARS* - 22 locations): The queue is a structure that can be used as a semaphore or mutex by the kernel.
-

III. Classes of misbehavior

The system behavior due to a fault can be classified as follows:

- **OK:** the system produces the same results as the golden execution.
- **Silent Data Corruption (SDC)**: according to [5] and others, when the output of the computation is altered, there is an SDC. The correctness is verified by providing a CRC code of the output values of the golden execution to be compared with the one produced by the single run.
- **Crash**: When a critical error occurs on the DUT, the system crashes, and the internal reset handler is called.
- **Freeze**: if a task does not end after the supposed deadline (defined through a time window of observation). This

time window is user-defined, and the reported experimental results refer to a dynamic time window defined as 50% more of the expected timing of the more extended task.
- **Delays**: if the execution presents no error, but the time exceeds the expected one, the difference can be interpreted as a missing deadline.

IV. Experimental Results

Experimental data have been gathered resorting to the EEMBC® Automotive suite [2] to select the benchmark tasks. The complete set of benchmarks comprises 16 applications and, among them, three models have been extensively tested:

- `a2time` (Angle to time conversion): it simulates a counter which measures the real-time delay between pulses sensed from a toothed wheel (gear) on the crankshaft of an engine.
- `tblook` (Table lookup and interpolation): it stores a limited amount of data pairs coming from one or more resources (sensors, connections, calculations) and interpolates missing pairs.
- `idctrn` (Inverse Discrete Cosine Transform): the Inverse Discrete Cosine Transform is applied to an input data-set representing a matrix of 64 bits values.

The setup is completed by two other dummy tasks that are performing operations on some hardware resource that has been treated as a shared one to stress the synchronization mechanism also for single-threaded reference benchmarks, i.e., the `a2time` and the `tblook`.

Injection campaigns have been carried out by injecting faults per location list. Their number follows the approach presented in [3] to obtain statistically significant results with an error margin of 1% and a confidence level of 95%.

A. Results Analysis

Since there was no evidence that SDCs have been produced among all experimental results, all further classifications do not include the SDC label for simplicity.

Looking more closely, the experimental results demonstrate that the resiliency of a system depends on the OS too. They clearly show how the FreeRTOS is affected by some vulnerabilities that lead to crashes. Figure 1 shows that the effects are statistically very similar among all benchmarks. This is not entirely unexpected because most fault locations are pointers to memory structures and indexes used to access and index elements of lists or vectors. Moreover, the low number of freezes probably relates to the injection location, making the system more efficient to crash instead of leaving it in an endless loop of unresponsive tasks altogether.

Figure 1 also suggests that the overall number of crashes and freezes does not depend on the benchmark, which is consistent with the fact that no internal memory of the tasks has been altered. On the contrary, delays are likely linked with the specificity of the benchmark, as the difference between `tblook` and the other two suggests. Still, on the broadside, it must be said that it also gives a wrong perspective among the several target groups, as Figure 2 is highlighting: due to the very different functional purposes of each group, the contribution in crashes, freezes, and delays vary a lot among them.

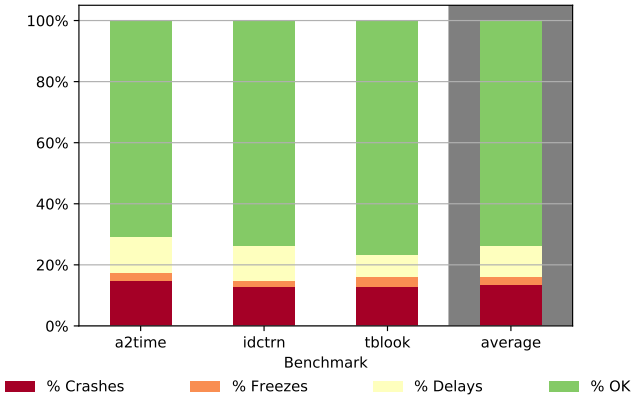

Fig. 1: Classification for all target locations by benchmarks

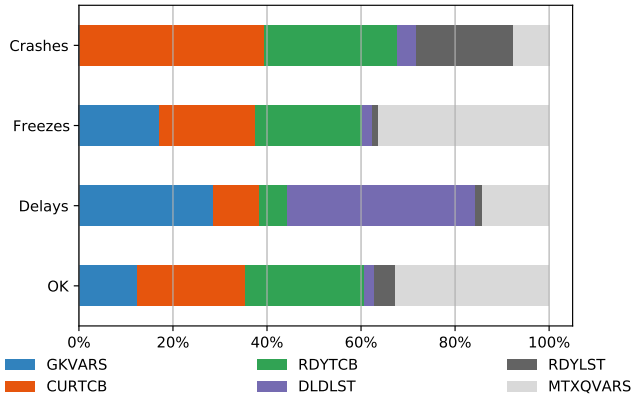

Fig. 2: Classification for all different target location groups

Figure 2 reports the classification across all six target location groups to appreciate the sensibility to the fault injection. It can be easily noticed how the `GKVARS` group does generate crashes, as `CURTCB` (almost 40%), `RDYTCB` and `RDYLST` do. Freezes are equally generated by alteration on any OS structure but the `RDYLST` and `DLDLST`. This is not forthcoming and probably relates to the high number of crashes when the `RDYLST` is corrupted and the even a higher number of delays produced by the `DLDLST` corruption. Regarding the delays, they also come from the `GKVARS` group, revealing a certain level of resiliency of the TCBs. When the corruption affects (parts of) the TCB that are not directly involved in the next cycle of scheduling, the outcome is still a properly running task but not necessarily the *expected* one. Eventually, the `MTXQVARS` group generates mainly freezes, delays, or no alteration at all, which confirms the accuracy of the tool since the `MTXQVARS` group includes all locations controlling mutex and semaphores.

V. Conclusions

This paper evaluates the dependability of FreeRTOS to SEU by building a Fault Injection Environment able to target data belonging to the operating system and trace the effects on the entire system. The results display the wide variety of impacts that FreeRTOS data structure disruption can lead to and allow us to suggest which are more sensitive to SEUs.

References

[1] Freertos. [Online]. Available: https://www.freertos.org/index.html

[2] EEMBC, *Autobench - Software benchmark data book*. www.eembc.org: EEMBC, 2015.

[3] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.

[4] G. S. Rodrigues and F. L. Kastensmidt, "Evaluating the behavior of successive approximation algorithms under soft errors," in *2017 18th IEEE Latin American Test Symposium (LATS)*, 2017, pp. 1–6.

[5] A. Vallero, A. Savino, A. Chatzidimitriou, M. Kaliorakis, M. Kooli, M. Riera, M. Anglada, G. Di Natale, A. Bosio, R. Canal, A. Gonzalez, D. Gizopoulos, R. Mariani, and S. Di Carlo, "SyRA: Early System Reliability Analysis for Cross-Layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 765–783, May 2019.

[6] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 460–469.