Fault Detection in Time- and Space-Partitioned Systems

Kleomar Almeida*, Ricardo C. Pinto, and José Rufino

University of Lisbon - Faculty of Sciences - LaSIGE
{kleomar,rcp}@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

Abstract. The next generation of space vehicles will integrate different mission functions on a shared computing platform using the advanced principle of Time and Space Partitioning (TSP). Improving the survivability of space vehicles requires reacting promptly on fault events, which implies timely fault detection.
This paper addresses the definition and design of fault-detection mechanisms for TSP hypervisors, covering both time and space domains. In spite of our focus in aerospace applications, the safety attributes and cost-effectiveness of TSP systems have a wider potential scope of applicability to other safety-critical environments, namely those involving autonomous vehicles in automotive, airborne and underwater applications.

Keywords: Fault Detection, Dependable Hypervisors, Time and Space Partitioning, Real-Time Systems, Mixed-Criticality

1 Introduction

Future space mission demand for innovative computing architectures and onboard software systems, meeting strict requisites of size, weight and power consumption (SWaP). To address SWaP requisites, functions which traditionally received dedicated resources are now integrated in a shared computing platform, using the advanced principle of *Time and Space Partitioning (TSP)*.

In TSP architectures the logical separation of applications in criticality domains, named *partitions* permits hosting several functions in the same computing platform, thus fulfilling the SWaP requirements. With partitioning one can achieve two main results: containment of faults in their domain of occurrence; enable independent software verification and validation, thus easing the mandatory certification process required by safety-critical applications.

* This work was partially supported: by the EC, through project IST-FP7-STREP-288195 (KARYON); by FCT/DAAD, through the transnational cooperation project PROPHECY; and by FCT, through the project PTDC/EEI-SCR/3200/2012 (READAPT), the Multiannual Funding Program, and the Individual Doctoral Grant SFRH/BD/72005/2010.

Although TSP systems conceptually guarantee that the partitions do not interfere with each other in the time and space domains, they do not guarantee *per se* that a partition may not exhibit faults - only that faults will not propagate to other partitions. Therefore effective fault detection mechanisms are needed to ensure correct operation on the overall system. Given the two dimensions of TSP, fault detection needs to be performed on both time and space domains.

This paper is organized as follows: Section 2 provides an overview of the AIR Technology and its components; Section 3 details the mechanisms for performing fault detection, both in the temporal and spatial domains; Section 4 concludes this paper and discusses future research work.

2 AIR Technology Overview

The **A**RINC 653 **I**n Space **R**eal-time operating system (AIR) technology was developed in an international consortium sponsored by European Space Agency (ESA) with the requirements of the aerospace industry in mind [9, 5]. Its scope, however, is not limited to aerospace but can be extended to domains with mixed-criticality applications, e.g. automotive [3, 6].

The AIR architecture was initially designed to implement and fulfill the TSP requirements of the ARINC 653 specification [1, 8]. It has since been improved, aiming at flexibility and offering features which are not covered by the ARINC 653 specification. An example of the improvements is the work being currently carried out to extend the AIR architecture to safely schedule applications over multiple processor cores [4].

2.1 System Architecture

The architecture of an AIR-based TSP system is depicted in Figure 1. The AIR architecture relies on the *Partition Management Kernel (PMK)*, a component supporting the whole system and enforcing robust TSP. Robust TSP ensures that partitions do not mutually interfere w.r.t. the fulfillment of real-time and addressing space encapsulation requirements. The PMK efficiently handles functionalities such as: partition scheduling and dispatching; low-level interrupt management; inter-partition communication support [9].

The AIR design foresees the possibility of hosting different Operating Systems (OSs) within different partitions, as is the case of Real-Time Operating Systems (RTOSs) and generic, non-real-time OSes. The Partition Operating System (POS) is encapsulated inside the *AIR POS Adaptation Layer (PAL)* allowing a more flexible integration of each POS.

The *APEX Interface* component provides a standard programming interface derived from the ARINC 653 specification [1], with the possibility of being subsetted and/or extended for certain partitions.

Additionally the AIR architecture contains a Health Monitor (HM) component responsible for monitoring components at the different layers of the architecture: hardware; system; partition; application layers. Upon the occurrence of

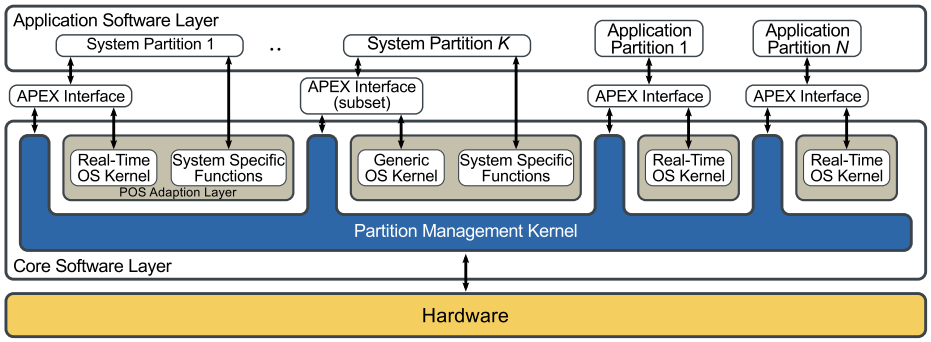


Fig. 1. AIR system architecture

an error (like a task deadline miss, memory protection violation, or even some hardware failure) an exception is raised. A handler provided by the application developer may be executed. This helps to confine errors within their domain of occurrence and prevent failures from propagating. The HM component is spread virtually in all AIR architecture components to ensure fault monitoring at all layers. Therefore the HM component does not appear in Figure 1.

2.2 Temporal Partitioning

Temporal partitioning is concerned with ensuring that the activities in one partition do not affect the timeliness of the activities in other partitions. Temporal partitioning is achieved through a two-level hierarchical scheduling scheme.

The first level corresponds to schedule partition execution. Partition execution scheduling is performed over a predetermined sequence of time windows. These windows are cyclically repeated over a *Major Time Frame (MTF)*. Partition activation corresponds to allocating one or more windows within the MTF. The order of activation is defined *off-line* at design time using appropriate tools, and a Partition Scheduling Table (PST) is created. The usage of PSTs provides deterministic execution, since partitions have allocated a predefined amount of time to access processor resources [9].

The second level corresponds to scheduling performed inside each partition. Tasks (processes, in the ARINC 653 terminology) are scheduled e.g., according to the native POS scheduler rules.

2.3 Spatial Partitioning

The other attribute of a TSP system is *spatial partitioning*. This attribute ensures that applications sharing the computing platform do not interfere with each other with regard to memory (code, data and execution context).

The enforcement of spatial partitioning provides guarantees regarding the isolation of partitions, allowing self-contained and safe application execution.

This enforcement is tightly coupled with the underlying computer architecture, needing additional mechanisms. The AIR technology exploits hardware-based memory protection mechanisms to provide spatial partitioning, and also spatial fault detection.

2.4 Health Monitoring

The AIR Health Monitor (HM) is responsible for detecting and handling hardware and software errors occurring at the different layers of an AIR-based system. An error - e.g., task deadline miss, memory protection violation, bound violation or hardware failure - is detected by the AIR HM entities. It is handled by a default Error Handler (EH) or one provided by the application developer.

As much as possible, the HM will confine the error propagation within its domain of occurrence: task level errors will cause an application error handler to be invoked; partition level errors trigger a response action defined by the Partition Health Monitor Table in the ARINC 653 configuration [1]. The response action may be shutting down the entire partition, reinitializing the partition again or simply ignoring the error (*see* Figure 2). Partition errors may stem from task level errors that cannot be handled by the application error handler. Errors detected at system level may lead the entire system to stop or reinitialize.

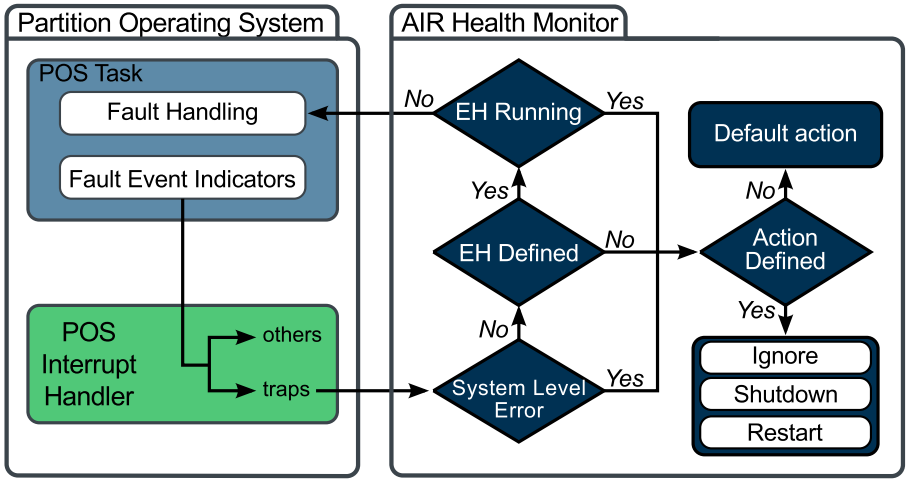


Fig. 2. AIR Health Monitoring scheme

3 Fault Detection in Time and Space Partitioning

The detection of faults on a TSP system must cover both domains: time and space. On the time domain, the detection of faults draws from the rigorous knowledge of the system timeliness. This knowledge is embodied by the PST, which

defines the partition schedule. On the spatial domain, it draws from AIR/ARINC 653 system configuration and the computer architecture mechanisms providing memory protection. In both cases the information pertaining to faults must be handed over to the Health Monitoring entity, which will trigger the correct handlers to deal with them.

3.1 Temporal Domain Fault Detection

An AIR-based system is composed by a set of partitions, where a partition (P_m) is defined as an application container. Each partition (P_m) contains a task set (τ_m), composed by individual tasks ($\tau_{m,q}$) holding the following properties:

- *Period* ($T_{m,q}$) - dictated by application requirements and/or environment;
- *Deadline* ($D_{m,q}$) - dictated by the application and/or environment requisites;
- *Capacity* ($C_{m,q}$) - obtained through WCET analysis of the task source code.

A temporal fault may occur due to a variety of factors. One of the factors may be the underestimation of the *Worst-Case Execution Time (WCET)* value during system design time. All estimations are susceptible to error, WCET analysis is no exception. In critical real-time systems the analysis cannot be underestimated, otherwise the result may be catastrophic. To avoid underestimation of the WCET analysis and to cope with the variance in computer programme execution time, the WCET analysis tools tend to overestimate the results (i.e., some pessimism is introduced into the analysis).

To ensure that all deadlines of a given partition task set are met, the system developer and/or integrator performs a *schedulability analysis* over the task set. This analysis makes use of: the estimated task capacity ($C_{m,q}$) through WCET analysis; a given scheduling algorithm; and the interference time of the other tasks in the task set and/or the other partitions. The result of the analysis is a computed *Worst-Case Response Time (WCRT)* for each task ($WCRT_{m,q}$). Given the nature of the analysis, two outcomes are possible: if any task $\tau_{m,q}$ does not fulfil the conditions of equation 1, the task set is deemed unschedulable; otherwise is deemed schedulable.

$$WCRT_{m,q} \leq D_{m,q} \leq T_{m,q} \tag{1}$$

When the task set is unschedulable, is the duty of the system developer and/or integrator to re-evaluate the temporal constraints of the given task set. In the case of the task set being schedulable, the WCRT analysis ensures that for each task $\tau_{m,q}$ of the task set, its execution will be finished before the given $WCRT_{m,q}$ for the respective task - based on the assumptions made by the system developer/integrator.

However, a change on the environment or in the assumptions made by the system developer/integrator or even an insufficient pessimism introduced during the WCET analysis may lead a task to violate its $WCRT_{m,q}$. Having this into account we propose the creation of a *Pseudo-Deadline* ($PD_{m,q}$) for each task of a given task set. The $PD_{m,q}$ is the trigger for a reactive mechanism that aims

at prompt detection and reaction over potential fault occurrences (task deadline miss). Its value stems from the $WCRT_{m,q}$ of the task, as depicted in Figure 3.

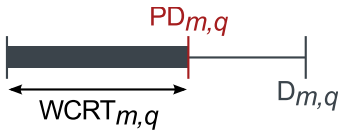


Fig. 3. Pseudo-deadline diagram

The relation between the $PD_{m,q}$ and $WCRT_{m,q}$ is represented in the equation 2, which shows how is the $PD_{m,q}$ value calculated.

$$\forall \tau_{m,q} \in \tau_m \rightarrow PD_{m,q} = WCRT_{m,q} \tag{2}$$

Analysis and Discussion

The advantages introduced by the $PD_{m,q}$ mechanism overcomes the disadvantages introduced. It possesses the following advantages: it permits to promptly react and detect the occurrence of faults; permits to improve the survivability of the space vehicles; and, no significant overhead w.r.t. the existent AIR task deadline violation monitoring mechanism [9].

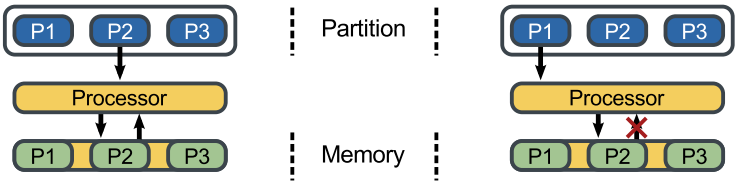
A disadvantage of the $PD_{m,q}$ is when the system is configured with a poor WCET analysis. This may lead to an erroneous placement of the $PD_{m,q}$ and also lead to an overload of warnings to the application coming from HM component, and therefore adding overhead to the whole system. This may occur when the $C_{m,q}$ of a task is underestimated, therefore the $PD_{m,q}$ could be constantly violated.

3.2 Spatial Domain Fault Detection

Another domain of segregation on a TSP system is the *address space*. On a shared computing platform, the space domain segregation implies that one application cannot access the addressing space of any other, either for reading or writing purposes. This partitioning is enforced by memory protection mechanisms.

The existence of memory protection mechanisms does not preclude the existence of faults - just the occurrence of errors. A misbehaving application may - intentionally or accidentally - try to access another application's addressing space. An example of an intentional address space violation is a rogue payload application in a dual-use satellite; an accidental violation is a wrongly valued pointer. In any scenario, the memory protection mechanisms would hinder the misbehaving application's intention, and raise an exception signalling such event.

An example is shown in Figure 4. On this example the Partition P2 accesses correctly its address space. Partition P1 tries to access the address space of P2, and the access is not validated.



(a) Correct access (c) Faulty access

Fig. 4. Memory access and protection mechanisms

The usage of memory management signalling mechanisms is the crux of both spatial domain fault detection and partitioning in TSP systems. An indication from these mechanisms is the signal that a space-domain fault occurred, which can then be dealt with by the appropriate functions, e.g. Health Monitoring.

Memory Protection Architectural Support

The enforcement of space partitioning must be supported by auxiliary hardware mechanisms offering memory protection. Any memory access involves the hardware mechanism, which validates such access. In recent computer architectures these mechanisms assume the shape of Memory Protection Unit (MPU), e.g. ARM Cortex-M3 [2] or Memory Management Unit (MMU), e.g. SPARC V8 [10] processors. Functionally, an MMU is an extended MPU.

The purpose of memory protection is to confine an application into an assigned region of the memory. The application is loaded into that region, and is only allowed to access that region, with any attempt to access another region being regarded as an exception. An example of this scheme is depicted in Figure 5, with Partitions being physically separated in memory regions.

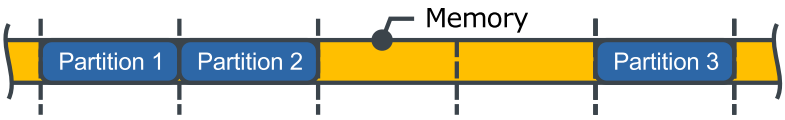


Fig. 5. Memory Segmentation

In the simplest case, a Partition is composed just by a compiled application. A compiled binary is composed by several sections: **code**, which must be immutable and contains the instructions to be executed; **data**, contains the (initialized) variables; **bss**, which contains zero-initialized and uninitialized global variables; **stack**, defined on a per task basis, which is used to store return information and local variables [7].

Such an organization raises issues w.r.t. memory protection, e.g. securing read-only sections. These are adressed by the hardware mechanisms, providing a set of permissions to protect memory areas - read, write and execute.

Memory Protection Fault Detection

A spatial domain fault (memory protection) can take two forms: an attempt to write to a read-only region; an attempt to access an area outside the application's segment. In any case an exception is raised by the MMU, which notifies the processor of the event.

In both cases, the auxiliary hardware mechanisms support fault detection. Upon an exception, the appropriate Health Monitoring mechanisms and handlers can be invoked to deal with the fault (see Figure 2). The information pertaining to the fault depends on the hardware architecture, e.g. the SPARC reference MMU [10] provides information regarding the offending address being accessed.

4 Conclusion and Future Work

This paper discusses fundamental ideas of how to achieve Fault Detection Mechanisms in Time- and Space-Partitioned systems. Our approach consists in enhancing TSP-based systems with the ability of promptly detect and react to the faults, thus improving the survivability of the space vehicles. The solution is performed differently at temporal and spatial domains. At the time domain we have to monitor application (processes) and partitions. At space domain we have to monitor the memory and input/output (e.g., memory mapped) spaces.

Our future work will be focused in the verification and validation of the proposed solution, and to provide the extension of the proposed mechanisms to introduce Proactive Fault Tolerance in the TSP-based systems.

References

1. Airlines Electronic Engineering Committee (AEEC): Avionics Application Software Standard Interface - ARINC 653 (Mar 2006)
2. ARM Limited: ARM v7-M Architecture Reference Manual (2010)
3. Consortium, A.: AUTOSAR: Specification of Operating System. Tech. rep., AUTOSAR Consortium (2006)
4. Craveiro, J., Rufino, J., Singhoff, F.: Architecture, mechanisms and scheduling analysis tool for multicore time- and space-partitioned systems. ACM SIGBED Review 8(3), 23–27 (Sep 2011)
5. ESA TSP Working Group: Avionics time and space partitioning user needs. Tech. Rep. 1, ESA/ESTEC, Technical Note TEC-SW/09-247/JW (2009)
6. Nóbrega Da Costa, P., Craveiro, J., Casimiro, A., Rufino, J.: Safety Kernel for Cooperative Sensor-Based Systems. In: Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS) (Sep 2013)
7. On-Line Applications Research: RTEMS C User's Guide. Tech. rep., OAR (2011)
8. Rufino, J., Filipe, S., Coutinho, M., Santos, S., Windsor, J.: ARINC 653 interface in RTEMS. In: Proc. of the Data Systems In Aerospace Conf. Naples, Italy (2007)
9. Rufino, J., Craveiro, J., Verissimo, P.: Architecting Robustness and Timeliness in a New Generation of Aerospace Systems. In: Casimiro, A., de Lemos, R., Gacek, C. (eds.) Architecting Dependable Systems VII, Lecture Notes in Computer Science, vol. 6420, pp. 146–170. Springer Berlin / Heidelberg (2010)
10. SPARC International Inc.: The SPARC Architecture Manual (1992)