Design and Development of a COTS-based Real-time Executive for Safety-critical Vehicular Applications

Luís Marques[1] and António Casimiro[2]

[1] FC/UL* lmarques@lasige.di.fc.ul.pt
[2] FC/UL casim@di.fc.ul.pt

Abstract. In this paper we describe some aspects of the work that was developed in the context of the HIDENETS European project, concerning the implementation of a real-time subsystem that is part of the adopted hybrid system architecture. This subsystem is responsible for ensuring the necessary predictability, which is required to implement safety-critical control applications. It provides a generic timing failure detection service and is able to execute a predefined safety task. We describe the architecture of this real-time executive and we point out some of the solutions that were employed to ensure a timely and predictable behavior, using the resources provided by the selected COTS hardware and development environment.

Abstract. Neste artigo descrevemos alguns aspectos do trabalho realizado no contexto do projecto Europeu HIDENETS, relativos à implementação de um subsistema de tempo-real que constitui uma das partes da arquitectura de sistema híbrida adoptada. Este subsistema é responsável por assegurar a previsibilidade necessária na construção de aplicações de controlo crítico em segurança. Fornece um serviço genérico de detecção de falta temporais e é capaz de executar uma tarefa pré-definida de segurança. Descrevemos a arquitectura deste executivo e apontamos algumas das soluções que foram empregues para assegurar um comportamento previsivel e atempado, usando os recursos disponibilizados pelo hardware COTS e pelo ambiente de desenvolvimento.

1 Introduction and context

The design and implementation of the real-time executive presented in this paper was performed as part of the HIDENETS European project [1]. This project aimed to develop and analyze end-to-end resilience solutions for distributed and mobile applications in ubiquitous communication scenarios, assuming highly dynamic, unreliable communication infrastructures. In the project, a safety-critical platooning application was selected for the demonstration of the developed solutions, as described in [2]. The real-time executive addressed here is a central part of the HIDENETS architecture and a fundamental element in the platooning application, to ensure the system safety.

A hybrid system architecture was adopted in HIDENETS, meaning that mobile nodes are composed of two different realms of operation, with respect to synchrony properties. The *payload part* of the system provides no guarantees about the timeliness of interactions taking place therein, but allows an unbounded number of payload processes to execute and

* FCUL. Navigators group: http://www.navigators.di.fc.ul.pt. This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

communicate with other nodes, namely through wireless networks. A second part, a *wormhole* subsystem, is a synchronous realm of operation, temporally isolated from the payload, in which a well-known set of real-time tasks execute.

This hybrid architecture provides several advantages over homogeneous ones. It allows developing applications (such as platooning) making use of complex algorithms without being limited by characteristics of traditional real-time designs, such as strict scheduling policies and the often difficult estimation of worst case execution times. It also allows the use of wireless communication and other data sources and communication channels characterized by high unpredictability. Yet, it still makes possible to implement safety critical applications, since the real-time part can be used to monitor the timeliness of these complex operations (through a Timely Timing Failure Detection (TTFD) service) and execute safety routines that are timely activated when timing faults occur in the payload, allowing for fail-safe or even fail-operational behaviors.

The objective of this paper is to highlight some aspects of this work, with a focus on the architectural aspects of the wormhole subsystem and on some of the specific solutions we employed to achieve a real-time behavior and the desired functionalities. The specific contributions of the paper are: 1) a description of the real-time executive architecture; 2) an explanation of the mechanisms for timing failure detection and safe behavior; 3) a brief discussion of implementation solutions.

2 Architecture of the real-time executive

Our real-time executive includes several functionalities, implemented as a set of real-time tasks which interact with the hardware through system library functions and tasks. Figure 1 illustrates the complete system architecture, including the payload part and its asynchronous control task, and the real-time part, with a number of tasks that we describe ahead, which are necessary to implement safety-critical applications such as platooning.

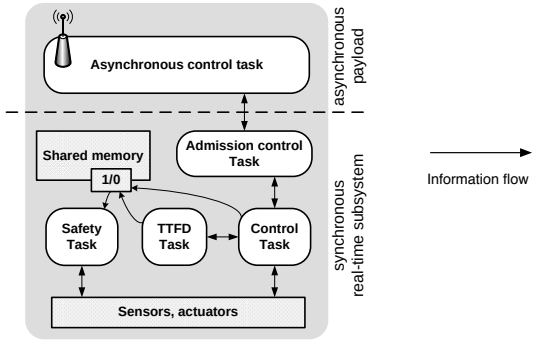

Fig. 1. System architecture

An admission control task handles the interface between the payload and the wormhole, processing payload commands and acting as an information gateway. Centralizing this responsibility allows the wormhole to be temporally isolated from the payload and to provide

a well-defined API. When the task is executed it only processes a limited number of requests pending on the incoming queue. This ensures a bounded execution time.

Payload requests are concerned with the TTFD service, with control commands that must be directed to the actuators, or with sensor reading operations. In the latter cases, the control task represented in the figure will in fact perform the interactions with the sensors/actuators on its own. A flag in shared memory is used to indicate if actuation commands received from the payload can be applied, or should be discarded. This flag is activated by the TTFD task when a timing fault occurs, and deactivated when some monitored action terminates on time. This mechanism is necessary to ensure the needed safety, by preventing that untimely payload control commands may be used.

This architecture tolerates payload timing faults, but no value faults are assumed in the fault model. Otherwise these would have to be handled through additional mechanisms, for instance through redundancy in the payload.

The TTFD service provides the following functions: `startTFD`, `stopTFD` and `restartTFD`. The `startTFD` function allows the specification of a deadline for some monitored timed action. The `stopTFD` function allows the indication that some timed action has finished and returns information on whether the action was timely terminated or not. If some monitoring activity is not terminated before the deadline, then a timing failure is detected and an handler is executed. In fact, this handler corresponds to the safety task, which is typically a control task with reduced functionality, but which is sufficient to allow the system to be fail-operational. A `restartTFD` function allows the atomic execution of a `stopTFD` request followed by a `startTFD` request (to ensure that there is always some action being monitored). All these functions are invoked by the admission control task.

3 Implementation

3.1 The hardware platform and development environment

The wormhole was built using a Lantronix's UDS100 device server, a hardware sold mainly to tunnel serial port data through an Ethernet connection. This device has limited computing power, having a 48MHz 8086 compatible CPU and only 256Kb of RAM. It provides one Ethernet and one RS-232 serial port. Despite these limitations it was deemed adequate for this proof-of-concept; better hardware can be used for more demanding real-world applications.

Being an 8086 CPU, the software is 16-bit, executing on 64 KB segments. The code was developed using Lantronix's SDK, which allows integration with its simple proprietary CoBox operating system, and Borland's Turbo C++ 5.02, the last version released supporting 16-bit development.

CoBox uses round robin cooperative multitasking, i.e. each task must yield control in a bounded amount of time, or a watchdog will restart the system. Execution starts in the task "ROOT", which transfers control to the customized code and (by default) launches several other system tasks, for network services, time management, configuration and setup. CoBox services four interrupts with the following decreasing priorities: serial interface, timer, network interface and standard. For reception of Ethernet frames, an interrupt handler removes the frames from the hardware ring buffer and places them in RAM. The "IP" system task later wakes up and inspects the buffer for inbound frames and handles them according to the Ethernet type field. Serial port I/O is performed directly by the tasks accessing the ports,

using system functions. Our admission control task does not perform busy wait operations on the serial port I/O. Instead it checks for the existence of requests, immediately yielding control if none is available. Otherwise, a limited number of requests is processed, to ensure a bounded execution time and to guarantee the temporal independence of the wormhole subsystem.

3.2 TTFD Service

The TTFD service is implemented by separating the timing failure detection and failure handler in two tasks. Whenever executed, the TTFD task is responsible for verifying if timing failures occured and setting the shared memory flag accordingly. The flag is always inspected by the control tasks (as explained above) and by the safety task. Therefore, either the payload control commands are processed and sent to actuators, or they are ignored and the safety task sends its own commands, based on local and simpler (more pessimistic) calculations. Note that the TTFD accuracy depends on the task release instant uncertainty, which is bounded by the worst-case round-robin cycle period.

The `startTFD` function uses a table to keep a list of monitored actions and their associated data (deadline), returning a handle to the entry if the request is accepted or an error indication if no entry is free. Clients must use this handle to call `stopTFD`, freeing the entry, and `restartTFD`, reusing it. Our implementation uses a compile-time constant to define the maximum number of entries (actions being monitored at a time); more entries imply more memory taken by the table and a larger worst-case execution time for starting and stopping monitoring calls.

The TTFD service is built using the local system clock. Deadline values are only meaningful with regard to the local wormhole and as such the payload must perform time arithmetic based on values previously acquired from the wormhole. It is also limited to the 1ms resolution of the hardware timer, which proved to be adequate for this application. The timing failure handler is guaranteed to execute during the interval $[t_d, t_d + \epsilon]$, where t_d is the deadline and ϵ the sum of the (bounded) worst case execution time of the wormhole tasks.

4 Concluding Remarks

In this short paper we motivated the need for a real-time executive in order to implement safety-critical applications, referring in particular to the HIDENETS architecture. The hybrid nature of this architecture allows for asynchronous and possibly complex control tasks to be used, and achieve improved control decisions (e.g., by using information received through wireless networks), while it also guarantees that a late control command will not be applied, but instead a safety control task will take over.

We described the internal architecture of this executive, explaining the fundamental interfaces and interactions, and we also briefly discussed some implementation aspects, essentially concerned with the TTFD service.

References

1. HIDENETS: Web site: http://www.hidenets.aau.dk/.
2. Marques, L., Casimiro, A., Calha, M.: Design and development of a proof-of-concept platooning application using the hidenets architecture. In: DSN '09: Proceedings of the International Conference on Dependable Systems and Networks. (2009) 223–228