Memory Protection in a Real-Time Kernel

Jorge Alçada*
University of Coimbra
Department of Informatics Engineering
Coimbra, Portugal
jfcosta@student.dei.uc.pt

Raul Barbosa and Johan Karlsson
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden
{rbarbosa, johan}@ce.chalmers.se

Abstract

This paper presents an extension of MicroC/OS-II to include MMU support used to provide memory partitioning while maintaining the kernel's real-time characteristics.

1. Introduction

Currently, a hot topic in embedded systems research is to develop techniques that allow several applications or subsystems to share a common hardware platform in a safe and reliable manner. The main objective for this research is to allow complex systems to be build from commercial-off-the-shelf (COTS) software components delivered by different vendors. Another important objective is to allow software applications of different criticality to execute on the same hardware. This way, the use of dedicated microcontrollers for specific subsystem can be avoided, which potentially reduces cost and improves reliability as it allows complex systems to be build using a smaller number of microcontrollers.

This trend is reflected in several industry standards such as the ARINC 653 specification [1], targeted for avionics systems, and the AUTOSAR initiative [2] for the automotive industry. These initiatives require a reliable method for memory partitioning, i.e. to run each application, as well as the operating system, in a protected memory address space.

While memory protection is an established feature of desktop and server computers, it is less frequently used in embedded real-time systems. One reason for this is that many microcontrollers lack the necessary hardware support. Another reason is that memory protection imposes time overhead and makes it more difficult to determine the worst-case execution time (WCET) of programs.

*This paper is based on Alçada's thesis for the degree of Licenciatura at the University of Coimbra, Portugal. This work was performed while he was an exchange student at Chalmers.

We are currently developing a prototype of a distributed real-time system, which we intend to use for experimentally assessing various techniques aimed at achieving the requirements of ARINC 653 and AUTOSAR, including approaches for building strong fault containment protection around software components. We use MicroC/OS-II [3] developed by Micrium as the operating systems for our prototype. We chose this kernel as it is freely available for academic purposes and because of its small memory footprint, excellent documentation support and well organized source code. These features make MicroC/OS-II easy to understand, to use and to extend. However, it lacks support for memory protection.

In this fast abstract, we describe how we have extended the MicroC/OS-II kernel to support memory protection. The extended version of the operating system runs on Freescale's MPC5554, which includes a Memory Management Unit (MMU). We demonstrate that it is possible to use the MMU without compromising the predictability of the execution time of the application programs, and we present an analysis of the time overhead imposed by the MMU.

The MPC5554 is a micro-controller based on the PowerPC architecture. Its processor core contains an MMU, and thus the processor does not have direct access to the main memory. Instead, the processor generates a virtual address that is translated by the MMU to the corresponding physical address. During this process of memory address translation, we can define access rights for each application process which are always checked by the MMU.

2. Design principles

The MPC5554's MMU offers memory management services, such as virtual memory and memory protection, and it comes with Translation Lookaside Buffer (TLB). The TLB is a small and very fast cache used to prevent large overheads in the address translation process. Each TLB entry contains the necessary information to manage the MMU, i.e. virtual and physical addresses, the size of each memory region and access rights.

MicroC/OS-II was extended to manage the MPC5554's MMU, as it is demonstrated in figure 1. The resulting kernel is now able to protect the memory address space of tasks. Each one of these tasks represent a *thread of execution*, and a group of threads that operate within the same memory region is referred to as *process*.

The application processes must be correctly divided in main memory so that the MMU can protect their memory region. Since in real-time systems all the processes reside in main memory, we used the linker to *statically* load each process to their corresponding memory address space. This process is done offline after compiling, thus avoiding further initialization overheads.

We also added support to run each thread in supervisor and/or user mode. Threads that run in user mode do not have direct access to the kernel and have limited view of the processor. Therefore, in order to be possible for these threads to communicate with the kernel, we implemented a *System Call* library. This library is a set of routines, globally shared, that acts as an interface for user applications to call the kernel's services.

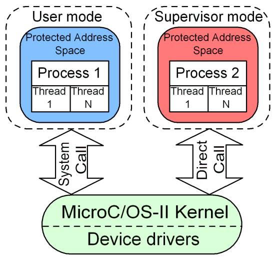

Figure 1. MicroC/OS-II extended architecture.

To guarantee that this extension would not affect the real-time characteristics of MicroC/OS-II, the design was implemented so it would not allow TLB cache misses. Thus, the TLB always contains the memory pages of the process that is currently running. Hence, the kernel updates the TLB in every process context switch (suspend the process that is currently running and give control of the processor to another process that is ready to run) [4].

3. Discussion

The memory protection mechanism added to MicroC/OS-II imposes a time overhead in every process context switch. In order to analyze exactly how much, we measured the time to perform a process context switch, varying the number of memory segments associated with the process that will run next. Since each memory segment requires at least one TLB entry and the MPC5554 MMU has a 32-entry TLB, this number can range from 0 to 32.

We constructed a test application composed of two processes that run on top of MicroC/OS-II. We measured the execution time from the first line of the context switch interrupt handler to the first instruction of process 2. These measurements were made with the MPC5554 running at a clock frequency of 136 MHz and with no compiler code optimizations. Figure 2 shows the process context switch time as a function of the number of memory segments.

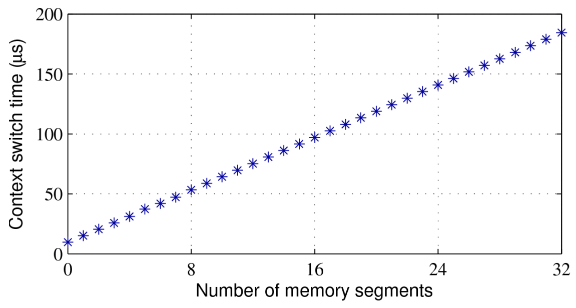

Figure 2. Context switch time measurements.

To verify that the TLB updates require a fixed time overhead, each configuration was measured several times. All the results were equal for the same number of memory segments. Therefore, the time to update the TLB is static and only depends on the number of process memory segments. Thus, we can take into account this time overhead for real-time scheduling analysis.

This result demonstrates that it is possible to use the MMU to provide memory protection without compromising real-time properties. Furthermore, the memory protection method implemented simplifies error detection and handling since it guarantees that there are no TLB access errors, unless there is a hardware or software fault.

On the other hand, we need to consider that, depending on execution time performance requirements, the overhead imposed by the MMU might not be acceptable. In addition, the paging system used by the MMU adds a small amount of internal fragmentation, because we are always restricted to the minimum page size available of 4 Kb.

References

[1] Draft 4 of Supplement 2 to ARINC Specification 653: Avionics Application Software Standard Interface, Part 1 - Required Services. August 2005.

[2] Harald Heinecke et al. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures.

[3] J. J. Labrosse. *MicroC/OS-II: The Real-Time Kernel*. Prentice-Hall, pub-PH:adr, 1992.

[4] Rune P. Anderson and Per Skarin. Memory Protection in a Real-Time Operating System. Master's thesis, Department of Automatic Control, Lund Institute of Technology, Nov. 2004.