ISSN : 0974 - 7435 Volume 10 Issue 15

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(15), 2014 [8857-8862]

Strategy of automobile electronic isolation protection mechanism based on embedded operating system

Hui Liu*, Yanliang Tan
Hengyang Normal University, Hengyang, 421002, (CHINA)

ABSTRACT

Strategy of automotive electronic isolation protection mechanism based on embedded operating system is mainly analyzed and discussed in this study. In combination with limited software mechanisms and hardware resources, isolation protection requirements in such aspects as application, operating system, interrupt service routine and task can be met and the control system memory access errors within a certain area are controlled so that possibility of embedded operating system failure is reduced.

KEYWORDS

Embedded operating system; Automotive electronics; Isolation protection mechanism; Strategy.

INTRODUCTION

In recent years, with the further upgrade of the complexity and the functions of the automobile electronic control system, there are 80 ECUs (Electronic Control Units) in an ordinary motor vehicle and five kinds of bus systems are adopted in them to realize the interaction and communication in these ECUs. Taking reliability, cost and other factors into consideration, it is an inevitable development trend to replace the large number of ECUs by a microcontroller which is more powerful in a small amount in today's automotive industry and previously there was a microcontroller operating in the proprietary ECUs [1]. If the integration of various functions in the ECU succeeds, then reliability and safety of automotive electronics in application need to be considered systematically. In the operation process of ECU, partition mechanism should be applied in the embedded software design of automotive electronics so as to isolate the various software components, effectively resist fault propagation and avoid interference among them. Based on this situation, an automobile electronic isolation protection mechanism is proposed in this paper. In this study, strategy of automotive electronic isolation protection mechanisms based on embedded operating system is mainly analyzed and discussed. In combination with limited software mechanisms and hardware resources, isolation protection requirements in such aspects as application, operating system, interrupt service routine and task can be met and memory access errors of the control system are controlled within a certain area so that possibility of embedded operating system failure is reduced.

Generally, software can be divided into application partition and operating system partition. Basic software modules like diagnostic communication software, operating system kernel, I/O control, storage software and peripherals are performed in a trusted operating system partition which is in the privileged mode. Logically, software components in application layer are divided in various applications that are in application partition which is in the unprivileged mode and can not be trusted and there are similarities among other operating systems and the applications. AUTO SAR Specification requires both isolation protection within application software and isolation protection between basic software and application software. To meet relevant security restrictions, isolation protection requirements in aspects as application, operating system and its execution are put forward in the specification on the basis of each segment of the program. The overall model of software partitioning in AUTO SAR structure is shown in Figure 1.

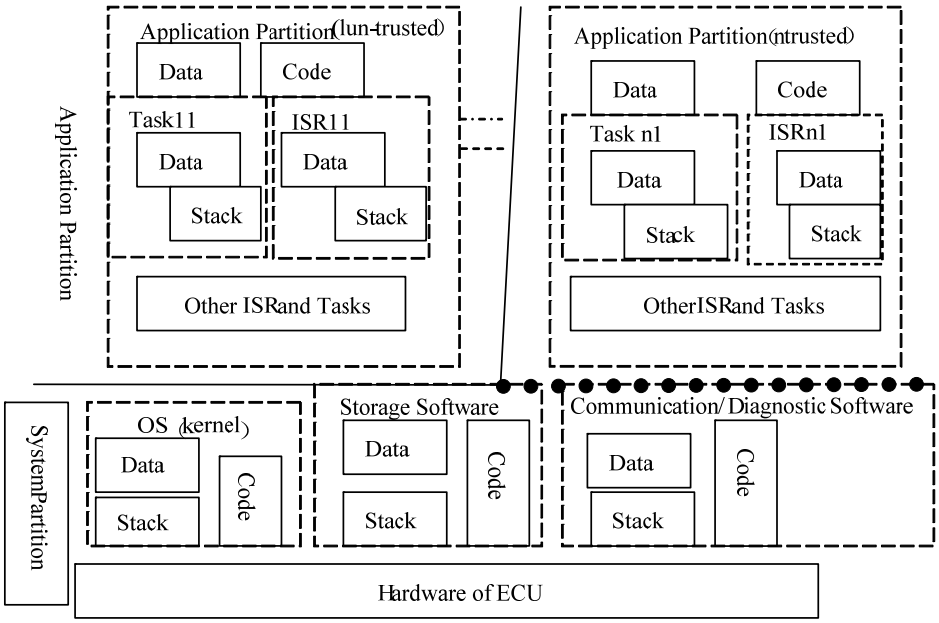

Figure 1 : Overall model of software partitioning in AUTO SAR architecture

(1) Protective operating system: Disable un-trusted applications to write in the stack and data segment of the system.

(2) Isolation of applications: a. prohibit un-trusted applications from reading or writing in the data segments of other applications; b. prohibit un-trusted application from writing in the IRS of other applications or data segments or stacks in tasks.

(3) Execution of isolation: Avoid writing of ISR of applications or tasks in stacks or data segments of ISR of un-trusted applications or tasks.

Only specific hardware can support the requirements of the above isolation protection. Due to the limitation of hardware resources in the embedded processor, it is often difficult to meet the requirements in the specification on isolation protection of some granularities and although it can be resolved by the MMU page fault exception handling mechanism, real-time response characteristics of the system are affected. As a combination of hardware and software mechanisms, the isolation protection mechanism put forward in this paper makes an effective use of the protective function of the hardware and enhances the safety of the automotive application system on the basis of software implementation [2].

INTRODUCTION OF OSEK OS STANDARD

The latest OSEK OS standard specifies the interfaces and objects that meet OSEK standard operating system must have, but do not prescribe specific implementation. As long as the concrete realization has a OSEK standard call interface and meet the OSEK standard features, it is called in line with OSEK standard operating system.

As an application-independent platform for application, OSEK operating system provides a single operating environment for microprocessor, which controls the operation of multiple real-time tasks concurrently. System call can also be used to manage the hardware resources. OSEK operating system is between the upper application and the underlying hardware; up to provide a standard interface for applications to call and down to control the underlying hardware. Operating of OSEK run is shown in Figure 2.

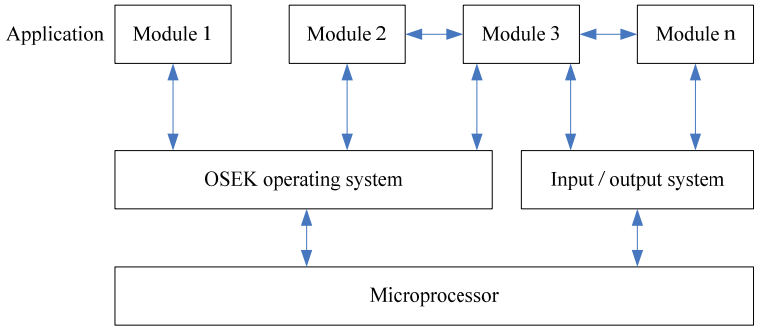

Figure 2 : Software Architecture of OSEK operating system

THE OVERALL FRAMEWORK AND FOUNDATION

The Overall Framework

Typically, there are three levels of isolation protection mechanism: a. The first level of isolation protection: Based on the operating modes of the processor, permission control of the operating system on memory access of applications can form an effective isolation between the application partition and the operating system partition; b. The second level of isolation protection: Automotive electronics conducts matching and examination of non operating system access by serial number of isolating in order to isolate different application partitions; c. The third level of isolation protection: It refers to the isolation of the executive bodies, the stack space of which in the isolation applications are used to effectively isolate the interior of the application partitions.

Realization of these three levels of isolation covers throughout exceptional handling in the operating system, system initialization and maintenance management. Figure 3 is framework of the multi-level isolation protection mechanism.

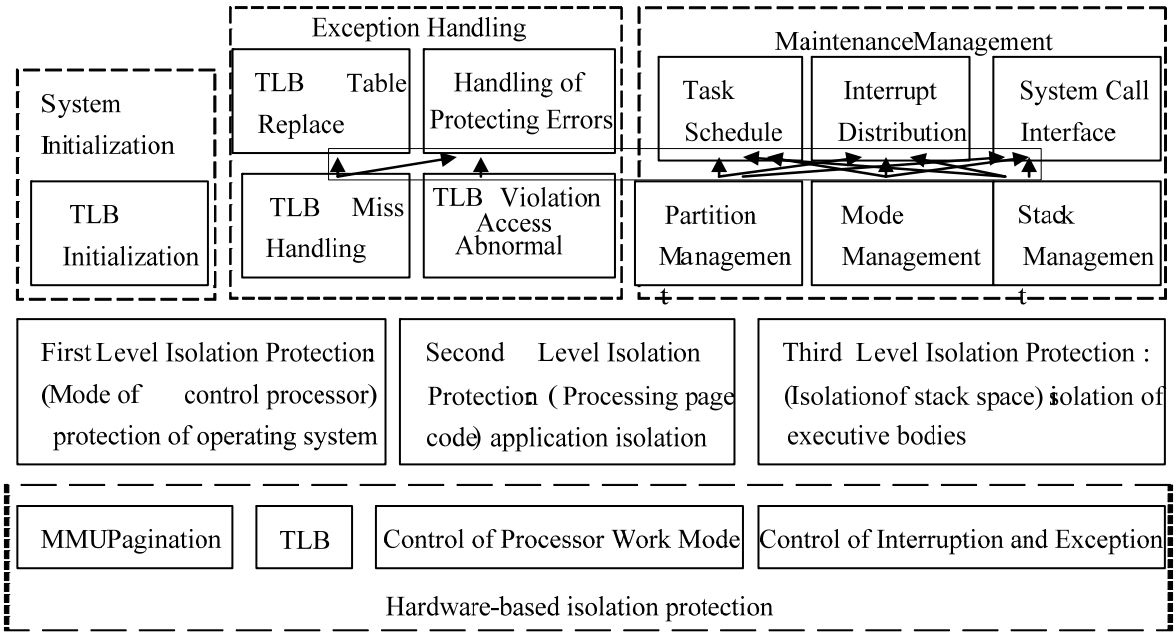

Figure 3 : Framework of multi-level isolation protection mechanism

Isolation Protection of the First and the Second Level and Hardware Requirements

Based on MPU or MMU hardware, both the first and the second level of isolation protection are within the category of pagination mechanism. The pagination mechanism of MPC563 4 processor is as the following: addresses generated in execution of instructions in conjunction with identification of MMU address space and the PID register brings forth page table entries of virtual address comparison which have recorded the corresponding actual page addresses, TID numbers and information of access permission. If they are a match, address translation is carried out and physical address eventually comes into being. However, there is also mismatching:

(1) No matching page address is found in the TLB. Abnormal TLB appears and page replacement needs to be conducted in exceptional handling procedure. If the matching page table entry of the page address is still not found after this, it indicates that the page address being visited does not exist, which will require error protection.

(2) There exists matching valid page address, but no matching TID in the entries of TLB and PID. Abnormal TLB shows up and the application of it in isolation of different application partitions can produce an application having access to other private application pages.

PROTECTION FAULT HANDLING AND MAINTENANCE MANAGEMENT

Compliance style of OSEK operating system

The diverse of application and different abilities (such as processors, storage space, etc.) of specific system require the operating systems also shows diversity, as well as facilitate understanding, research and implemented a part collection of properties of OSEK operating system and increase the OSEK operating system can be cut, OSEK standard specifies four compliance categories, the realization of a category can be called a version of the OSEK operating system:

(1) BCC1, only supports basic tasks, one for each priority task, different tasks have different priorities, does not support multiple activation;

(2) ECC1, on the basis of BCC1 support event synchronization and extended tasks;

(3) BCC2, on the basis of BCC1 support multiple activation and multiple tasks in each priority

(4) ECC2, on the basis of ECC1 supports activation multiple tasks and each of a plurality of priorities.

These four categories are in line with the upward-compatible: Any application for BCCx comply with operating system development classes can be transplanted without modification to ECCx operating system, any applications for xCC1 meet the class operating system can transplant to xCC2 operating system without modify, as shown in Figure 4.

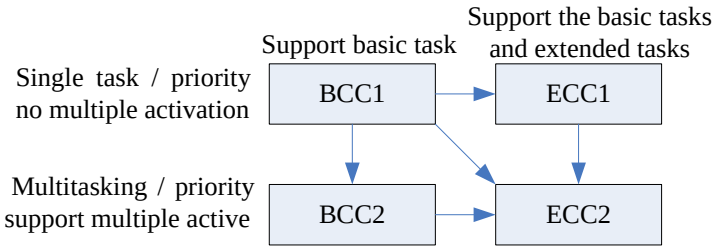

Figure 4 : Comply category of OSEK operating system

Control Flow in the System and its Impact

During the actual operating process of the system, such conditions as interruption, call functions and task switching caused by task scheduling happen in the operating system because of application. Thereby, control flow in the system will frequently convert between the actual operating system and execution bodies of application [3]. In each conversion, the operating system can ensure validity of stack conversion, operating mode of processor and the partitioning of applications. To avoid omissions, the focus of this study is to realize interrupt distributed processing in the system and to handle calls and switch tasks through program design and analysis based on aspect oriented programming.

Interrupt Distributed Processing

In essence, interrupt distributed processing can achieve the conversion between the targeted ISR and the interrupt execution body. The processor will automatically convert the system into privileged mode at the time of the interruption. The interrupt distributed processing procedure is able to implement appropriate treatment in accordance with the properties of the targeted ISR and the interrupted execution body. Together system services, un-trusted or trusted ISR, un-trusted or trusted task constitute interrupted execution body while un-trusted or trusted ISR ISR composes the targeted ISR. The specific processes for the interrupt distributed program are as the following:

(1) Preprocessing of the user ISR calls. The implementation of the application partition switching, mode switching and stack switching is determined by the application partition and credibility of the targeted ISR. A. If there is a difference between the application partition number of the PID register and that of the targeted ISR, PID will be protected in the interrupt context and new application partition number will be set for PID. B. If the targeted ISR is trusted, stack switching

and mode switching will not be implemented. C. If the targeted ISR is un-trusted, the user mode will be switched by the privileged mode and stack of the interrupt system will be switched by the user stack of the targeted ISR.

(2) Saving of the interrupt context. It should be ensured that the interrupt context can be saved in a trusted stack: interrupt context can saved directly without switching of stacks when there is interruption in the ISR, trusted tasks and system services; when there is interruption in ISR and un-trusted tasks, stack switching is carried out and the interrupt context can be saved in the terminal system or stacks of the operating system [4].

(3) The following actions need to be taken in the returning of the user ISR call: a. If it returns through trusted ISR, the user mode will be switched to the privileged mode; meanwhile, ISR user stack will be switched by interrupt system stack; b. if it returns by un-trusted ISR, stack switching and mode switching will be conducted timely and it will return directly to the interrupt distributed processing.

(4) There are mainly three kinds of possibilities in the post-processing of interrupt distributed processing: a. Return to the previous ISR. If there is still interrupt nesting, the interrupt context will be restored and returns to the user ISR program; b. Return to the interrupted task. If there is no interrupt nesting, stack of the task system stack will be switched. When there is no need to re-schedule the task and the result is still in the state of the original interrupted task, the interrupt context should be recovered from the task system and returns to the original position where the task is interrupted; c. Switch to a new task. If the scheduling result is to implement switching of other tasks, task switching function should ensure the switching of application partitions, mode switching and stack switching.

Calling Interface of the Operating System

Fundamentally, calling interface of the operating system is in fact the conversion between operating system servicer and the caller. System caller can be divided into two kinds, the trusted and the un-trusted. In the execution of system services and system calling process, there is no need for switching of application partition. System call handling is only to be concerned about the stack space and changes of processor modes.

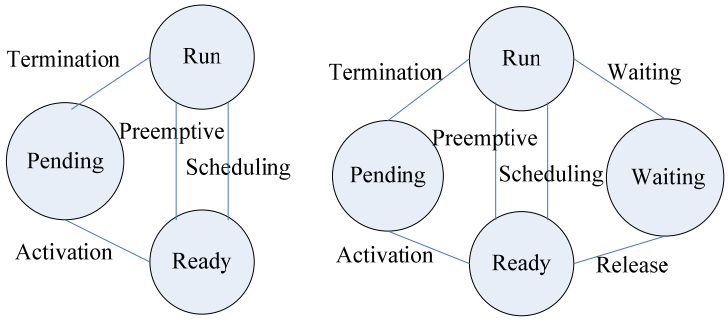

Figure 5 : State transition diagram of basic tasks and extended task

(1) When the system calls trusted ISR or tasks, the processor itself is in privileged mode and there is no need for stack switching and mode switching. System services can directly run in the stack of the caller through function call.

(2) When the system calls un-trusted ISR or tasks, user mode should be switched to privileged mode and ISR or the user stack of the task is switched to the system stack[5]. In the execution of this instruction, the processor automatically switches to privileged mode and goes to the exception handler. Therefore, the exception handler can complete stack switching. After the completion of system service function, the stack is switched to the original user stack while returning to the exception handler and mode switching is automatically completed by the abnormal return instruction.

CONCLUSION

In a word, there is isolation protection mechanism within an embedded operating system. Incorporation of various software components of security and integrity from different channels can reduce the difficulty in making the software system meet the high security level so ad to meet the growing demand for security of automotive electronic control and isolation protection mechanism is challenging for the automotive electronic embedded system with real-time requirements but limited hardware resources. In this study, the effective integration of software and hardware resources can fundamentally reduce the demand for hardware resources, improve automobile hardware performance and meet the demand for integrity and high security level of automotive control systems.

ACKNOWLEDGEMENT

The research supported by Natural Science Foundation of China（Grant No.11375058）, China Hunan Provincial Science & Technology Department project（Grant No.2014FJ3061）, China Hunan Provincial Hengyang Administration of Science &Technology project（Grant No.2014KJ23）

REFERENCES

[1] Sheng-Lin Gui, Lei Luo, Sen-Sen Tang, Yang Meng; Optimal Static Partition Configuration in ARINC653 System, Journal of Electronic Science and Technology, **4**,177-178 **(2011)**.

[2] Deng Jun, Li Hong, Fang Zheng, Luo Dan, Hu Qi; Improvement and Implementation of Storage Protection for AUTOSAR OS, Chinese Journal of Scientific Instrument, **9**, 164-166 **(2011)**.

[3] S.L.Gui, L.Luo, S.S.Tang, et al; Optimal static partition configuration in ARINC653 system, Journal of Electronic Science and Technology of China, **9(4)**, 373-378 **(2011)**.

[4] Chen Lirong, Yan Liming, Luo Lei, An Isolation and Protection Mechanism of Automotive Electronic Embedded Operating System, Journal of University of Electronic Science and Technology of China, **3**, 154-155 **(2014)**.

[5] Zhang Lvhong; Design and Implementation of Smart OSEK OS4.0 Based on AUTOSAR, Hangzhou: Zhejiang University, 188-189 **(2010).**