6G-04

Yet Another Target Software Simulator

Rui Shao† Yukikazu Nakamoto†

Graduate School of Applied Informatics, University of Hyogo†

1 Introduction

Our lives are surrounded by computer systems, embedded systems, such as computer systems in automotive, information appliances, smart phones, and smart houses. These embedded systems become more complex to realize richer services. Meanwhile, shorter development period of embedded systems is required due to market pressure. To solve the problem, engineers develop commonly embedded software with simulators.

There is an important problem, however, that when engineers develop these embedded systems, they develop the simulation program by rewriting existing program of target machine with PC as a standard, and naturally this process is troublesome and it would often encounter problems of compatibility.

Therefore, this paper addresses how we solve these problems with automotive software as a target and addresses that we intend to design a simulator program that could be executed on PC and it can simulate the environment of Autosar operating system which is used for automotive computer.

2 Purpose

This paper proposes a new approach to do the simulations of Autosar OS[1] on PC with rewriting as small programs as possible on top of Autosar OS classic. That is, we design a simulator program which can simulate the function of Autosar OS such as the tasks and the interrupt service routines feature and this program is able to use at both target machine (ARM Rx650) and PC environment (x86).

We intend to design this simulator program as an alternative to the Autosar OS when engineers want to write control programs with their PC with C language. During this simulation, they can also do the model in the loop simulation (MILS[2]) by using our simulator program with the software MATLAB/Simulink. With using MATLAB/Simulink, engineers can simulate the whole process as with the real plant machine. When they finish writing the programs, they can be executed as the real Autosar operating system do, such as the task function, the interruption mechanism.

We implement this simulator on top of Linux. The design and implementation are very limited function of task and interrupt processing in Autosar classic.

Through this work, we understand functions and behaviors of Autosar OS classic and Linux as another purpose.

3 Related Works

In our research, we can also use the CPU simulation to simulate the CPU with target plants with our host machine. By using CPU simulation, the simulator program we write can be more like the target machine operating system which means Autosar OS. There are, however, problems such as slow execution for target programs. We developed V850 CPU simulator using QEMU, a translation-based CPU simulator [3]. In this experience, a Dhrystone MIPS is calculated as approximately 45 MIPS with the current PC with 2.2 GHz Pentium DualCore.

Our research is to simulate that how tasks execute on Autosar OS, and as we know Autosar is a standardized extension of OSEK. Therefore, its API is aligned with OSEK/VDX OS. Trampoline[4] is also a static RTOS for small embedded systems. Its API is aligned with OSEK/VDX OS and AUTOSAR OS 4.1 standards which means the API in Trampoline can also be used on Autosar OS.

4 Implementing Task with Thread

The design and implementation of the OS simulator are very limited function of task and interrupt processing as mentioned before. We briefly summarized functions of tasks of Autosar classic and pthread, which are used to implement the task functions. A task in Autosar OS is mapped to a Linux thread[5]. TABLE 1 shows a rough mapping between Autosar APIs and Linux pthread APIs. In this paper, Autosar API names are based on [6].

TABLE 1 Mapping between tasks and pthread

	Autosar OS	Linux
Task creation	Configuration	`pthread_create`
Task start	`ActivateTask`	
Task priority set	Configuration	`pthread_setschedparam`
Task termination	`Terminate-Task`	`pthread_exit`

To simulate `ActivateTask` and `Terminate-Task`, we can write roughly a function ActiveTask using `ptherad_create` and `pthread_exit`. However, task creation in AutoSar is done at configuration time while pthread is created in

execution time. Moreover, setting a priority of tasks are set on task creation time while a priority of a thread can be done after thread creation. We use a real-time pthread with FIFO parameter to continue the thread execution by its end. pthread for task have lower priorities because higher priority are reserved for interrupt processing as mention below. So we design the task creation at start-up time when creating thread, setting priority are done and thread are locked with mutex to make a task pool. We name this API `ConfigCreateTask()`. In `ConfigCreate-Task`, a thread is create and waited while a thread priority is set with mutex. `ActivateTask` picks up a task from the task pool by `pthread_mutex_nlock`(). `TerminateTask` pushes back a task to the task pool by `pthread_mutex_lock()`. Thus, we designed this program that we use pthread function to simulate the task feature in Autosar OS. To localize difference between

5 Implementing interrupt processing with signal

A type is an interrupt is restricted to “C2ISR” in this simulator because C2ISR is managed by the OS. An interrupt and interrupt processing in Autosar OS is mapped to a Linux signal and thread, respectively. TABLE 2 shows a rough mapping between these relations.

TABLE 2 Mapping between interrupts and signal

	Autosar OS	Linux
Interrupt	Interrupt	signal
Interrupt processing	C2ISR	thread
Interrupt masking and unmaskign	`SuspendOS-Interrupts` `ResumeOS-Interrupt`	`sigprocmask`

An interrupt processing routine is implemented by a thread. In this simulation, both of priority of tasks and interrupt processing are implemented with pthread priority. Higher priorities in pthread are assigned to the interrupt processing and lower priorities are assigned to the task. We use a real-time signal because the number of user-define signal is limited in an ordinal signal. We create a thread corresponding to an interrupt handler ahead and assign a thread priority with an interrupt priority to make an interrupt handler pool like a task pool in an interrupt configuration. We name this API `ConfigISR()`. When an interrupt comes in, the thread corresponding to the interrupt is unlocked. Nested execution of an interrupt handler is available. Masking and unmasking of interrupt, which are `SuspendOSInterrupts` and `ResumeOS-Interrupt` in AutosarOS, can also be implemented by `sigprocmask`. As we knew these method, we designed this program that we use signal, param.sched_priority and mutex function to simulate the interruption feature in Autosar OS.

6 Conclusions and Future Work

We presented the concept that how we design this simulator program to simulate the Autosar operating system with Linux. We presented how we solve the problem with a new way that we wrote a simulator program which can simulate the target operating system without rewriting the target program and we present the difference of tasks and interruption feature between Autosar OS and Linux.

In our research, there is still another problem that the host simulator just simulate the work in Autosar OS That is a execution time difference between the host OS and AutosarOS that is the calculation speed of CPUs are different. We intend to insert a delay function into the end of the simulator that to make program execution time of the host machine as same as that in target machine.

Acknowledgement

This work is partly supported by JSPS KAKENHI Grant Numbers 16H02800 and Fujitsu Ten Limited.

Reference

[1] S. Furst, et al., “AUTOSAR - A Worldwide Standard is on the Road,” Proc. 14th International VDI Congress Electronic Systems for Vehicles, 2009.

[2] G. Jain, “Memory Models for Embedded Multicore Architecture” in *Real World Multicore Embedded Systems*: Chapter 4. Newnes, 2013.

[3] Y. Nakamoto, et al., "Proposing A Hybrid Software Execution Environment for Distributed Embedded Systems", Proc. 13th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing Workshops, pp.176-183, 2010.

[4] Main Page of Trampoline. "https://github.com/TrampolineRTOS/trampoline".

[5] Robert Love, “Linux Kernel Development”, 3rd ed, Addison Wesley, 2011.

[6] Center for Embedded Computing Systems Graduate School of Information Science, Nagoya Univ.,“RTOS External Specification for Next Generation Automotive Systems”, 2014. (in Japanese).