·体系结构与软件技术· 文章编号:1000-3428(2017)01-0079-07 文献标志码:A 中图分类号:TP311

AUTOSAR OS 存储保护机制的形式化验证框架

李 青,朱晓冉,郭 建

(华东师范大学 计算机科学与软件工程学院,上海 200062)

摘 要: 传统汽车标准存储模块的安全性较低,汽车电子操作系统在访问存储模块时会出现访问越界和数据冲突等问题。为此,提出一种操作系统的存储保护机制。运用进程代数给出满足存储保护机制的形式化验证框架,从逻辑上讨论 AUTOSAR 存储保护机制的重要性,使用进程代数方法对该机制建立形式化模型,并根据 AUTOSAR 规范,抽取无死锁性、安全性、活性等性质,运用模型检验工具 PAT 实现该模型,并对各个存储模块的读写访问性质进行验证。仿真结果表明,与传统的汽车标准相比,该机制符合 AUTOSAR OS 规范,具有较高的安全性。

关键词: 操作系统;存储保护;进程代数;PAT 工具;形式化验证

中文引用格式: 李 青,朱晓冉,郭 建. AUTOSAR OS 存储保护机制的形式化验证框架[J]. 计算机工程,2017,43(1):79-85.

英文引用格式: Li Qing, Zhu Xiaoran, Guo Jian. Formal Verification Framework for AUTOSAR OS Storage Protection Mechanism[J]. Computer Engineering, 2017, 43(1):79-85.

Formal Verification Framework for AUTOSAR OS Storage Protection Mechanism

LI Qing, ZHU Xiaoran, GUO Jian

(School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

【Abstract】 Since the security of standard storage module of traditional vehicle is relatively low, problems like accessing out of bounds or data conflicting may occur while the electronic operating system of vehicles accesses the storage modules. So a storage protection mechanism of operation systems is proposed. The process algebra is used to give the formal verification framework that ensures the storage protection mechanism, and the importance of the AUTOSAR storage protection mechanism from the logical point of view is discussed. A formal model of the mechanism is built using process algebra, and the properties like deadlock freeness, safety and liveness are extracted according to the AUTOSAR specifications. Then the model is implemented using PAT——a modelling and verification tool for process algebra, and the correctness of read-write properties of each storage module is checked. Simulation result shows that compared with conventional car standard, the mechanism conforms with the AUTOSAR OS specification and has higher security.

【Key words】 operating system; storage protection; process algebra; PAT tool; formal verification

DOI: 10.3969/j.issn.1000-3428.2017.01.014

0 概述

存储保护机制通过隔离系统中的各个区域,可以有效地避免故障传播问题,同时,也可以使得系统调试更加简单。因此,存储保护机制被广泛应用于复杂度高、安全性要求苛刻的嵌入式实时操作系统中。

随着汽车工业的发展,汽车电子的软硬件系统变得越来越复杂,存储保护机制的应用对汽车电子操作系统显得尤为重要。目前被广泛应用的用于汽车电子、带有接口的开放式系统(Open Systems and the Corresponding Interfaces for Automotive Electronics,OSEK)规范没有提供系统运行时隔离各个应用程序的机制,无法满足汽车电子操作系统的高安全性。2005 年提出的 AUTOSAR 架构建立了一个标准化的系统架构方案,不仅兼容 OSEK/VDX 提供

基金项目: 国家自然科学基金中丹合作项目(6136113600);国家自然科学基金重点项目(61532019);"核高基"重大专项(2014ZX01038-101-001)。

作者简介: 李 青(1991—),男,硕士研究生,主研方向为嵌入式系统建模与验证;朱晓冉,博士研究生;郭 建,副教授。

收稿日期: 2015-12-15 **修回日期:** 2016-02-10 **E-mail:** 51131500020@ecnu.cn

的接口,同时增加了时间保护、服务端保护以及存储保护等机制,存储保护通过分区机制来防止操作系统各部件的干扰,进一步确保了汽车电子操作系统的安全性[1-3]。

基于 AUTOSAR 规范的汽车电子操作系统中的存储保护机制要求在应用程序访问存储模块之前,对该应用程序进行检测,确定其权限和访问地址安全无误后才能访问各个存储模块中的数据。存储保护机制可以有效地防止应用程序访问超过其管理权限的系统存储模块。并且,加载了存储保护机制的汽车电子操作系统在访问出错时,可以返回应用程序中出错的位置及原因,使得技术人员调试更加方便、准确。存储保护机制在操作系统的访问越界和数据冲突等问题上起着重要的作用,因此,对 AUTOSAR 下的存储保护机制的分析至关重要[4]。

形式化方法应用在嵌入式系统安全性验证方面,可提高复杂系统的可靠性。相比于大规模测试,它从逻辑上验证了系统的正确性。许多技术人员选择使用形式化方法对汽车电子基础软件如操作系统内核(Kernel)[5]、通信协议[6]等方面进行验证,以确保其安全性。

文献[7]使用符号模型检测技术从设计层面对 AUTOSAR 规范下的 FlexRay 进行形式化分析,确保了系统的设计需求满足相应规范。文献[8]使用基于形式化模型的测试用例生成方法,针对 AUTOSAR 规范中定义的系统服务生成了覆盖较为完全的测试用例,为 AUTOSAR 下汽车电子操作系统的测试工作奠定了重要基础。

对于存储保护问题,相关学者做了一定的形式化分析与验证。文献[9]使用 Promela 建模语言,利用模型检测工具 SPIN,对 L4 微内核操作系统中的内存保护机制进行形式化分析,证明了内存管理策略的正确性和有效性。文献[10]使用定理证明、模型检测等方法,验证和分析了 Android 操作系统中内存管理机制的系统进程、用户交互进程等方面相关属性,为开发人员更好地了解 Android 操作系统内存管理机制提供了理论依据。

针对 AUTOSAR 规范汽车电子操作系统下的内存保护机制,本文提出运用 CSP(Communicating Sequential Processes)语言建立通过管理内存权限来防止地址越界、数据冲突的形式化模型,提取内存保护机制方面满足 AUTOSAR 规范需求的性质,并使用模型检测工具 PAT 验证这些性质。

1 预备知识

1.1 进程代数

进程代数通过对进程的行为、特性及并发交互进行建模,从而可以对这些进程间的通信、并发等情况进行推导及验证。通过这个过程来发现对象在运行过程中可能出现的问题。进程代数中相互之间传递消息是通过通道传输的[11],进程代数可以应用于许多场景中,不仅仅是计算机软件,也可以对通信协议或者硬件进行建模与推导[12]。

与其他形式语言一样,进程代数对对象的主要特性进行抽象,并以对象间的交互为切入点,以分析对象间的具体运行情况。进程代数是一种描述传递消息、接收消息的并发的代数系统的符号。

进程代数是一种形式化的建模语言,其语法为:

$$P,Q ::= \mathrm{stop} \mid \mathrm{skip} \mid a \rightarrow P \mid P;Q \mid P[]Q \mid$$
$$c!\ a \rightarrow P \mid c?\ a \rightarrow P \mid P \mid\mid Q \mid P \mid\mid\mid Q \mid$$
$$\mathrm{if}\ a(b)\ \{P\}\ \mathrm{else}\ \{Q\} \mid \mu X \cdot F(X)$$

其中,$c!\ a \rightarrow P$ 表示通过通道 c 发送 a 信息之后执行进程 P;$c?\ a \rightarrow P$ 表示通过通道 c 接收 x 信息之后执行进程 P;$P[]Q$ 表示外部选择;$P;Q$ 表示进程 P 与进程 Q 之间顺序执行;$P \mid\mid Q$ 表示进程 P 与进程 Q 并发执行;$P \mid\mid\mid Q$表示进程 P 与进程 Q 交替执行;if $a(b)\{P\}$ else $\{Q\}$表示如果动作 a 的状态是 b 那么执行 P 进程,否则执行 Q 进程;$\mu X \cdot F(X)$表示求解 $F(X)=X$ 的不动点方程。

1.2 AUTOSAR OS 存储保护机制

存储保护是 AUTOSAR 规范中的一个重要保护机制,一旦存储模块的堆栈、代码等数据被错误的权限程序访问,可能会造成数据篡改、数据覆盖等问题。AUTOSAR 规范中定义的存储保护访问对象包括应用程序、任务和中断服务子例程,应用程序是一个包含了任务、中断服务子例程、报警器、调度表、计数器的集合[13-15]。

对于可信的应用程序,操作系统允许其在禁用监控和保护功能的情况下运行。它们可以直接访问内存、操作系统、API,在运行时也不会被中断,并且可以在特权模式下运行。反之,操作系统不允许非可信应用程序在禁用监控和保护功能的情况下运行,并且它们不可以直接访问内存、操作系统、API,在运行时会中断。

存储保护机制如图 1 所示,首先通过应用程序、任务和中断服务子例程请求 OS 内核通过内存保护机制访问数据,然后 OS 内核得到指令传递存储地址到存储模块中,存储模块得到指令返回后访问数据给 OS 内核,最后将数据返回应用程序、任务和中断服务子例程。

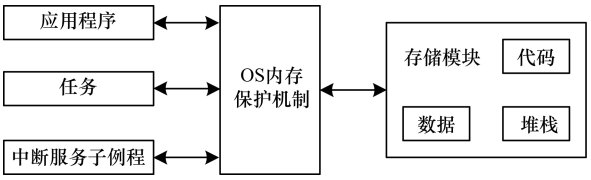

图 1　存储保护通信机制

在基于 AUTOSAR 的操作系统中有 3 个对象需要保护:堆栈,代码,数据。

堆栈包括任务堆栈和中断堆栈。一般堆栈是各自独立的,不会与其他任务和中断服务子例程共享。数据包括任务和中断数据,私有数据可以被自身程序访问,公有数据可以被所有程序访问,代码包括了私有代码仅仅可以被自身程序访问,以及共享代码可以被所有应用程序、任务、中断服务子例程访问。

操作系统访问程序分成可信的访问程序和非可信的访问程序,应用程序、任务、中断服务子例程通过堆栈保护、代码保护、数据保护等保护机制访问代码、堆栈、数据时会发生异常。

应用程序被操作系统调用时存在 3 种状态:

1)当应用程序处于运行状态时表示操作系统可以调用应用程序。

2)当应用程序处于就绪状态时表示操作系统只有在调用激活函数后才可以调用应用程序。

3)当应用程序处于阻塞状态时表示操作系统完全不可以调用应用程序。

这里特别说明本文中应用程序是指一个包含了任务、中断服务子例程的集合。

应用程序、任务以及中断服务子例程通过 OS 内存保护机制传输访问地址和访问权限,并判断地址是否越界和权限的问题,确认地址合法和权限正确后,可以进行读写访问,对于地址越界和无权限访问一旦出错则会进入 AUTOSAR 的错误处理机制。

2　AUTOSAR OS 内存形式化建模

在内存保护架构中,首先了解内存当中逻辑上包括了几个存储块,即应用数据模块、中断数据模块、任务数据模块、中断堆栈模块、任务堆栈模块、代码模块以及外围设备模块。

可以分成 3 个大的模块:一个模块有应用程序访问的应用数据模块、代码模块以及外围设备模块;另一个模块有中断服务子例程访问的中断数据模块、中断堆栈模块;最后一个模块有任务访问的任务数据模块、任务堆栈模块。

2.1　内存保护架构

本文对内存保护机制进行分析,具体研究应用程序访问、中断服务子例程访问、任务程序访问在内存中的运行过程。

非可信的应用程序不可以读写访问私有数据模块,而可信的应用程序可以读写访问私有数据模块,可信和非可信的应用程序都可以读写访问共享数据模块。非可信的应用程序不可以读写访问私有代码模块,而可信的应用程序可以读写访问私有代码模块,可信和非可信的应用程序都可以读写访问共享代码模块。

非可信和可信的应用程序都可以读写访问外围设备,不过非可信的应用程序最好不要读访问外围设备,如图 2 所示。

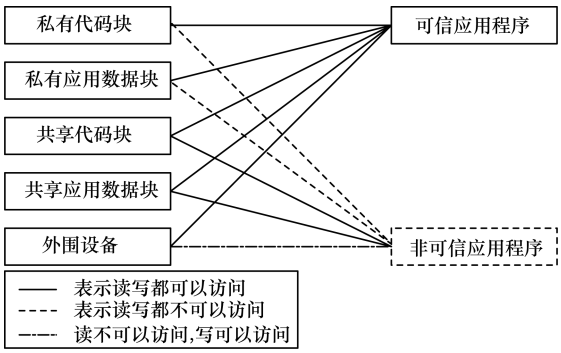

图 2　可信与非可信的应用程序访问示意图

非可信的中断服务子例程不可以读写访问私有数据模块,而可信的中断服务子例程可以读写访问私有数据模块,可信和非可信的中断服务子例程都可以读写访问共享数据模块。

非可信的中断服务子例程不可以读写访问私有堆栈模块,而可信的中断服务子例程可以读写访问私有堆栈模块,如图 3 所示。

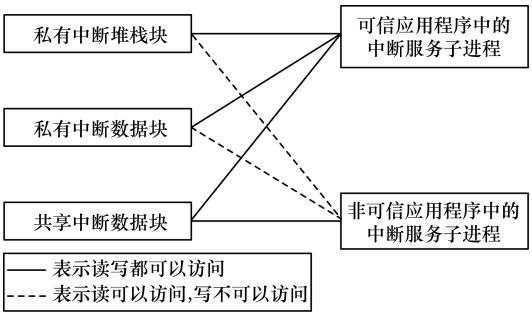

图 3　可信与非可信的中断服务子例程访问示意图

非可信的任务不可以读写访问私有数据模块,而可信的任务可以读写访问私有数据模块,可信和非可信的任务都可以读写访问共享数据模块。

非可信的任务不可以读写访问私有堆栈模块,而可信的任务可以读写访问私有堆栈模块,如图 4 所示。

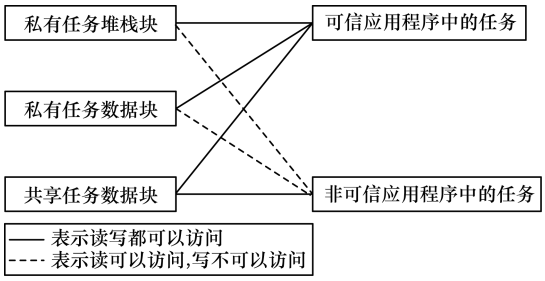

图 4 可信与非可信的任务访问示意图

2.2 总体建模

对系统建模,系统主要分成读和写 2 个进程并发 *sys = read||write*,读操作和写操作模型分别包括了应用程序、任务、中断服务子例程如图 5 所示,这里把读操作模型和写操作模型分开是因为读操作模型和写操作模型在具体对于某个存储模块时操作的权限是不相同的,有些程序可以对读操作访问,对写操作不能访问;有些则相反对写操作访问,对读操作不能访问,具体分析根据 AUTOSAR 内存保护规范在内存保护结构中已经说明。

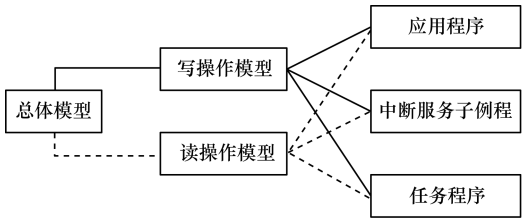

图 5 总体建模

2.3 读操作模型

建立读操作模型需要内存地址的基本变量、通道和一些访问控制权限的变量:

1)列出一些基本常量,内存里面分成多个区域,每个区域的地址范围由私有地址和共享地址组成,在私有地址和共享范围地址中分成了首地址和尾地址,例如 *appdata_addpri_start* 表示应用程序私有数据模块的首地址,*appdata_addpri_end* 表示应用程序私有数据模块的尾地址,*appdata_addlib_start* 表示应用程序共享数据模块的首地址,*appdata_addlib_end* 表示应用程序共享数据模块的尾地址,其余的(外围设备模块、代码模块、中断数据模块、中断堆栈模块、任务数据模块、任务堆栈模块)同样也有私有数据模块的首地址、私有数据模块的尾地址、共享数据模块的首地址、共享数据模块的尾地址。

2)定义一些通道用于消息传输和数据传输。包括从应用程序到 OS 内存保护机制,任务到 OS 内存保护机制,中断服务子例程到 OS 内存保护机制,OS 内存保护机制到应用数据模块,OS 内存保护机制到外围设备模块,OS 内存保护机制到代码模块,OS 内存保护机制到中断数据模块,OS 内存保护机制到中断堆栈模块,OS 内存保护机制到任务数据模块,OS 内存保护机制到任务堆栈模块,数据传输和消息传输共享同一个通道,对于读进程来说,*ch_osr* 表示应用程序、任务、中断服务子例程通过 *ch_osr* 通道发送给系统管理者访问地址和访问权限以及最后接收从系统管理者发送回来的读写数据,*osr_appdatar*, *osr_peripheralr*, *osr_coder*, *osr_isrdatar*, *osr_isrstackr*,*osr_taskdatar*,*osr_taskstackr* 表示系统管理者通过这些通道发送从 *ch_osr* 通道接收过来的访问地址和访问权限,写进程与其类似同样也定义了8 个通道。

3)定义变量,*powr_error*,*poww_error* 分别表示读写访问错误时返回的布尔值,*pror_to_os_address*, *prow_to_os_address* 表示读写访问的地址,*pror_to_os_flag*, *prow_to_os_flag* 表示读写访问的权限,*os_to_pror_data*,*os_to_prow_data* 表示读写访问从各个存储模块中返回的数据,*Map* 表示内存模块。

读进程包括了 pror 进程、checkr 进程以及 memr 进程并发的 *read = pror||checkr||memr*。

pror 进程用来发送读指令访问数据的地址和权限。

```
ch_osr! pror_to_os_address. pror_to_os_flag- > ch_osr?
x1- > pror;
```

checkr 进程有 2 个作用:首先用于检查存储模块的数据是否有重叠(前一个存储模块的尾地址不能大于后一个存储模块的首地址),一旦有重叠则说明存储器本身地址表示范围存在问题,不能调用读写访问指令。

```
if (appdata_addpri_end < appdata_addlib_start) {osr};
```

另外在判断访问数据的地址是否符合规范,权限是否符合规范,访问地址和权限只要其中一个出现错误就进入错误访问机制,返回出错信息,如果都正确的话,那么返回正确数据给任务和中断服务子例程,其模型为:

```
osr = ch_osr? x1_1. x1_2- >
if( ( ( (x1_1 > = appdata_addpri_start) &&
(x1_1 < = appdata_addpri_end) ) &&
(x1_2 < = 1) ) || ( (x1_1 > = appdata_addlib_start) &&
(x1_1 < = appdata_addlib_end) ) )
{errr|powr_error = true;} - >
osr_appdatar! pror_to_os_address. pror_to_os_flag- >
osr_appdatar? x2- > ch_osr! os_to_pror_data- > osr}
else {Skip};
```

memr 用来管理存储模块中的各个模块的数据,包括了 7 个数据模块:应用数据模块,外围设备模块,代码模块,中断数据模块,中断堆栈模块,任务数据模块,任务堆栈模块。

```
memr = appdatar||peripheralr||coder||isrdatar||isrstackr|
```

|taskdatar||taskstackr;

appdatar 用来接收从 OS 内存保护机制传输过来正确的访问地址和访问权限,然后传输给 OS 内存保护机制访问数据值。

```
appdatar = osr_appdatar? x2_1. x2_2- >
ar{os_to_pror_data = map[x2_1];}- >
osr_appdatar! os_to_pror_data- > appdatar;
```

2.4　写操作模型

写进程同样也包含了 prow 进程、checkw 进程、以及 memw 进程。

```
write = prow||checkw||memw;
```

prow 进程与 pror 进程基本一样用来写访问数据的地址和权限。

```
prow = ch_osw! prow_to_os_address. pror_to_os_flag- >
ch_osw? x11- > prow;
```

checkw 进程和 checkr 进程相同,检查存储模块的地址是否有重叠,一旦重叠则退出程序。当地址为合法地址时,那么跳转到 osw 进程,在 osw 进程中,根据规范对于读写进程有几个地方处理不同,同时建模过程的处理也不同:1)需求规范中对于私有堆栈模块,非可信的任务和中断服务子例程是不可以写访问私有堆栈模块的,但是却可以读访问私有堆栈模块。2)需求规范中对于私有数据模块,非可信的任务和中断服务子例程是不可以写访问私有数据模块的,但是却可以读访问私有数据模块。3)需求规范中对于外围设备模块,非可信的应用程序是不可以读访问外围设备模块的,但是却可以写访问外围设备模块,其模型为:

```
osw = ch_osw? x11_1. x11_2- >
if ((((x11_1 > = code_addpri_start) &&
(x11_1 < = code_addpri_end)) &&(x11_2 < = 3)) ||
((x11_1 > = code_addlib_start) &&
(x11_1 < = code_addlib_end)))
{errw{poww_error = true;}- >
c14! prow_to_os_address. prow_to_os_flag- >
c14? x14- > ch_osw! os_to_prow_data- > osw}
else{Skip};
```

memw 用来管理存储模块中的各个模块的数据,其功能与 memr 类似。

```
memw = appdataw||peripheralw||codew||isrdataw||
isrstackw||taskdataw||taskstackw;
```

codew 用来接收从 OS 内存保护机制传输过来正确的访问地址和访问权限,然后找到数据,修改数据,写回存储块中。

```
codew = c14? x14_1. x14_2- >
cw{os_to_prow_data = map[x14_1] + 10;}- >
c14! os_to_prow_data- > codew;
```

3　性质与验证

根据 AUTOSAR 规范提取了内存保护机制中的一些性质包括无死锁性、安全性、活性。这些性质可以用线性时态逻辑(Linear Temporal Logic,LTL)表示和断言表示。线性时态逻辑是基于一阶逻辑,增加时态算子,可刻画系统的时间特性的逻辑,其语法为:

$$\Phi ::= \Phi \wedge \Psi \mid \Phi \vee \Psi \mid F(\Phi) \mid G(\Phi) \mid X(\Phi) \mid (\Phi U \Psi) \mid (\Phi W \Psi) \mid (\Phi R \Psi)$$

其中,$F(\Phi)$表示在将来某个时刻满足性质 Φ,用符号◇表示;$G(\Phi)$表示在所有时刻都满足性质 Φ,用符号□表示;$X(\Phi)$表示在下一个时刻满足性质 Φ,用符号○表示;$\Phi U \Psi$ 表示开始满足性质 Φ 直到某个时刻满足性质 Ψ。$\Phi W \Psi$ 表示 2 种情况:1)一直满足性质 Φ 直到 Ψ 的出现;2)开始同时满足性质 Φ 和 Ψ,直到某个时刻不满足性质 Φ;$(\Phi R \Psi)$表示第 1)种情况一直满足性质 Ψ,第 2)种情况开始满足性质 Φ,直到某个时刻同时满足性质 Ψ。

3.1　无死锁性

死锁是系统中多个进程同时需要占用共享资源,而导致进程无法继续前进的情况,AUTOSAR 存储管理机制要求进程在任何时候都可访问资源,而不会出现死锁。

```
#assert sys() deadlockfree;
```

3.2　安全性

应用数据安全性(Application Data Safety,ADS):可信的应用程序可以读写访问私有数据模块,非可信的应用程序不可以读写访问私有数据模块。

```
#assert pror() |= [](ch_osr. 5. 0-> <> ch_osr. 1)
||(ch_osr. 5. 1-> <> ch_osr. 1);
#assert prow() |= [](ch_osw. 5. 0-> <> ch_osw. 11)
||(ch_osw. 5. 1-> <> ch_osw. 11);
```

应用程序的读进程通过 *ch_osr* 通道访问应用程序数据模块中的私有地址 5,数字 0 表示可信权限,数字 1 表示私有权限,若访问权限正确,则最终会返回应用程序数据模块的数据 1,写进程类似。

应用外围设备安全性(Application Peripheral Safety,APS):可信的应用程序可以读写访问外围设备模块,非可信的应用程序可以写访问外围设备模块。

```
#assert pror() |= [](ch_osr. 15. 0-> <> ch_osr. 2)
||(ch_osr. 15. 1-> <> ch_osr. 2);
#assert prow() |= [](ch_osw. 15. 2-> <> ch_osw. 12);
```

应用程序的读进程通过 *ch_osr* 通道访问外围设备模块中的私有地址 15,若访问权限正确,则最终会返回外围设备模块的数据 2,写进程类似。

应用代码安全性(Application Code Safety,ACS):可信的应用程序可以读写访问私有和共享代码模块,非可信的应用程序可以读写访问共享代码模块。

```
#assert pror() |= [](ch_osr.25.0-> <>ch_osr.3)
||(ch_osr.25.2-> <>ch_osr.3);
#assert prow() |= [](ch_osw.25.0-> <>ch_osw.13)
||(ch_osw.25.1-> <>ch_osw.13)
||(ch_osw.25.2-> <>ch_osw.13);
```

应用程序的读进程通过 *ch_osr* 通道访问代码模块中的私有地址 25,其中数字 2 表示共享权限,若访问权限正确,则最终会返回代码模块的数据 3,写进程类似。

中断服务子例程数据安全性(Interrupt Service Routine Data Safety,IDS):可信的中断服务子例程可以读写访问私有数据模块,非可信的中断服务子例程不可以读写访问私有数据模块。

```
#assert pror() |= [](ch_osr.45.0-> <>ch_osr.4)
||(ch_osr.45.1-> <>ch_osr.4);
#assert prow() |= [](ch_osw.45.0-> <>ch_osw.14)
||(ch_osw.45.1-> <>ch_osw.14);
```

中断服务子例程的读进程通过 *ch_osr* 通道访问中断数据模块中的私有地址 45,若访问权限正确,则最终会返回中断数据模块的数据 4,写进程类似。

中断服务子例程堆栈安全性(Interrupt Service Routine Stack Safety,ISS):可信的中断服务子例程可以读写访问私有堆栈模块,非可信的中断服务子例程不可以读写访问私有堆栈模块。

```
#assert pror() |= [](ch_osr.55.0-> <>ch_osr.5)
||(ch_osr.55.1-> <>ch_osr.5);
#assert prow() |= [](ch_osw.55.0-> <>ch_osw.15)
||(ch_osw.55.1-> <>ch_osw.15);
```

中断服务子例程的读进程通过 *ch_osr* 通道访问中断堆栈模块中的私有地址 55,若访问权限正确,则最终会返回中断堆栈模块的数据 5,写进程类似。

任务数据安全性(Task Data Safety,TDS):可信的任务可以读写访问私有数据模块,非可信的任务可以不读写访问私有数据模块。

```
#assert pror() |= [](ch_osr.75.0-> <>ch_osr.6)
||(ch_osr.75.1-> <>ch_osr.6);
#assert prow() |= [](ch_osw.75.0-> <>ch_osw.16)
||(ch_osw.75.1-> <>ch_osw.16);
```

任务的读进程通过 *ch_osr* 通道访问任务数据模块中的私有地址 75,若访问权限正确,则最终会返回任务数据模块的数据 6,写进程类似。

任务堆栈安全性(Task Stack Safety,TSS):可信的任务可以读访问私有堆栈模块,非可信的任务不可以读写访问私有堆栈模块。

```
#assert pror() |= [](ch_osr.85.0-> <>ch_osr.7)
||(ch_osr.85.1-> <>ch_osr.7);
#assert prow() |= [](ch_osw.85.0-> <>ch_osw.17)
||(ch_osw.85.1-> <>ch_osw.17);
```

任务的读进程通过 *ch_osr* 通道访问任务堆栈模块中的私有地址,若访问权限正确,则最终会返回任务堆栈模块的数据 7,写进程类似。

3.3 活性

读进程活性(Read Active,RA):根据 AUTOSAR 规范,同一时间至少有一个程序可以对存储模块进行读访问。

```
#define function1(os_to_pror_data = =1)
||(os_to_pror_data = =2)
||(os_to_pror_data = =3)
||(os_to_pror_data = =4)
||(os_to_pror_data = =5)
||(os_to_pror_data = =6)
||(os_to_pror_data = =7);
#assert read() reaches function1;
```

读进程访问的范围限制在 1 ~7 这 7 个值中,任何不等于这 7 个值的访问都是错误访问,错误访问会调用错误函数。

写进程活性(Write Active,WA):对于写访问来说同一时间有且仅有一个程序对存储模块进行写访问。以下是对其中私有数据模块进行写访问的验证:

```
#define function2 (os_to_prow_data = =11)
&&(os_to_prow_data! =12)
&&(os_to_prow_data! =13)
&&(os_to_prow_data! =14)
&&(os_to_prow_data! =15)
&&(os_to_prow_data! =16)
&&(os_to_prow_data! =17);
#assert write() reaches function2;
```

写进程访问的范围限制在 11 ~17 这 7 个值中,任何不等于这 7 个值的访问都是错误访问,错误访问会调用错误函数。

3.4 结果分析

为了验证提出的线性时态逻辑,选用新加坡国立大学的 PAT 工具,建立其 CSP 模型,并通过仿真、模型验证等方法检验模型的正确性。

在 PAT 中可以验证上节中提取的性质是否满足模型,从而说明模型是否符合 AUTOSAR OS 的内存保护规范,性质的验证结果如图 6、表 1 所示。

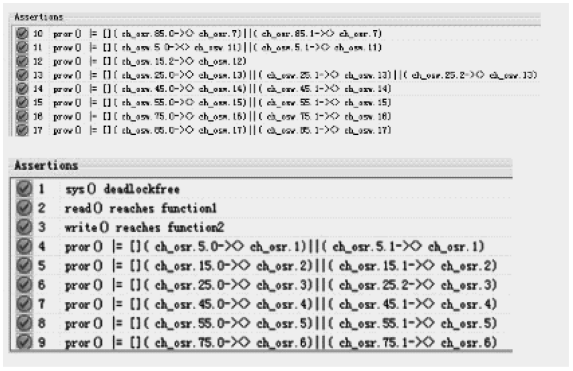

图 6 PAT 中的运行结果

表 1　结果验证与分析

性质类别	性质	结果
无死锁性	DL	正确
安全性	ADS	正确
	APS	正确
	ACS	正确
	IDS	正确
	ISS	正确
	TDS	正确
	TSS	正确
活性	RA	正确
	WA	正确

根据表 1 中所列的结果可知,建立的内存模型满足所提取的性质。

通过对无死锁性验证,保证了系统中应用程序、任务、中断服务子例程同时需要访问某个数据模块时不会产生死锁的情况。安全性要求应用程序、任务、中断服务子例程访问各个存储模块时,只有在读写访问权限正确的情况下,读写进程才能返回访问数据,否则,不会返回访问数据,并且调用错误函数。活性验证性质 RA 和 WA 的验证结果表明,在同一时刻对一个共享存储模块访问时,只能有一个读进程或者一个写进程进行访问,等到其访问结束返回数据之后,其他才能请求 OS 访问存储模块。

4　结束语

本文运用进程代数对 AUTOSAR OS 的存储保护机制进行形式化建模,并根据规范提取了 10 个性质,在验证工具 PAT 上实现了该模型,并对模型所提取的性质进行验证,结果表明,提出的存储保护模型框架符合 AUTOSAR OS 规范。本文主要研究单核 AUTOSAR OS 的存储保护问题,由于 AUTOSAR 规范支持多核操作系统,但是多核下的存储保护过程较为复杂,如何实现多核下的存储保护将是下一阶段的主要工作。

参考文献

[1] 项　晨. 参照 AUTOSAR 标准的存储器驱动模块的研究与实现[D]. 上海:复旦大学,2011.

[2] Zeng Haibo, Natale M D. Efficient Implementation of AUTOSAR Components with Minimal Memory Usage[C]// Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems. Washington D. C., USA: IEEE Press,2012:130-137.

[3] Ferrari A, Natale M D, Gentile G, et al. Time and Memory Tradeoffs in the Implementation of AUTOSAR Components[C]//Proceedings of Design, Automation & Test in Europe Conference & Exhibition. New York, USA:ACM Press,2009:864-869.

[4] 邓　俊,李　红,方　正,等. AUTOSAR OS 存储保护方案的改进与实现[J]. 仪器仪表学报,2011,32(9):2146-2152.

[5] Peng Yunhui, Huang Yanhong, Su Ting, et al. Modeling and Verification of AUTOSAR OS and EMS Application[C]// Proceedings of IEEE International Symposium on Theoretical Aspects of Software Engineering. Washington D. C., USA:IEEE Press,2013:37-44.

[6] Ran Qinwen, Wu Xi, Li Xin, et al. Modeling and Verifying the TTCAN Protocol Using Timed CSP[C]// Proceedings of IEEE International Symposium on Theoretical Aspects of Software Engineering. Washington D. C., USA:IEEE Press,2014:90-97.

[7] Bahig G, El-Kadi A, Salem A. Formal Verification of AUTOSAR FlexRay State Manager[C]//Proceedings of the 9th International Design & Test Symposium. Washington D. C., USA:IEEE Computer Society,2014: 193-198.

[8] Fang Ling, Kitamura T. Formal Model-based Test for AUTOSAR Multicore RTOS[C]//Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation. Washington D. C., USA: IEEE Computer Society,2012:251-259.

[9] 陈超超,曾庆凯. 采用 SPIN 的 L4 内存管理形式化验证[J]. 计算机工程,2009,35(11):131-133.

[10] 郭婷婷. Android 操作系统内存管理形式化分析[D]. 上海:华东师范大学,2013.

[11] 孙　麒,张云华. 基于 CSP 的形式化方法研究[J]. 浙江理工大学学报,2009,26(4):557-560.

[12] 周巢尘. 通信的顺序进程及其研究[J]. 计算机学报,1983(1):3-11.

[13] 钱振江. 安全操作系统形式化设计与验证方法研究[D]. 南京:南京大学,2013.

[14] 吴立军,骆翔宇,陈清亮. 基于动态内存和状态管理的模型检测新方法[J]. 计算机科学,2011,38(11):191-195.

[15] 顾　飞. 共享内存 IPC 机制的形式化验证与实现[D]. 兰州:兰州大学,2012.

编辑　刘　冰