软件安全核的可信性问题*)

黎忠文[1] 姚绍文[2]

(厦门大学计算机与信息工程学院 厦门 361005)[1] (云南大学软件学院 昆明 650091)[2]

摘 要 软件的大量应用,使控制系统面临严峻的安全考验,陷入了安全危机中,迫切需要新的安全保障技术。安全核就是应运而生的一种安全保障新概念,其可信性直接关系到安全核的有效性和系统的安危。面对安全核可信性问题,测试和限制安全核尺寸是当前采用的方法,它们极大地制约了安全核技术在复杂系统中的应用。本文分析了安全核可信性的本质;结合安全关键系统的基本构架,提出了从安全需求分析开始到安全核生成过程中,如何通过形式化的方法来提高安全核可信性的方法,为安全核技术在复杂系统中的应用提供了一种新思路;以交通灯控制为例全过程地实现和验证了所提出思想的正确性和可行性。

关键词 安全核,可信性,安全策略,安全关键系统,形式化原理,交通灯

The Dependability Problem of Software Safety Kernel

LI Zhong-Wen[1] YAO Shao-Wen[2]

(Computer and Information Engineering College, Xiamen University, Xiamen 365001)[1]

(School of Software, Yunnan University, Kunming 650091)[2]

Abstract Since software is being used in the control system largely, the possibility of serious damage resulting from a software defect is considerable and growing, and then the control systems are plunging into safety crisis. In fact, control systems are in urgent need of new safety assurance technologies. Safety kernel is a new concept of safety assurance. It emerges, as the time requires. Of course, its dependability plays an important role on system. The present methods to improve the dependability of safety kernel are to limit its size and test. These methods prevent largely safety kernel from being used in complex systems. This paper at first analyzes comprehensively correctness requirements of safety kernel and the basis frame of safety-critical system. Then, based on all of these, a method that how to construct and improve the dependability of safety kernel is put forward from the safety requirements. Also This paper is a demonstration that safety kernel is a feasible and desirable technique for software in complex safety-critical systems. At last, taking the control system for the traffic lights as example, the whole process and its correctness have been achieved and proved.

Keywords Safety kernel, Dependability, Safety policy, Safety-critical systems, Formal method, Traffic light

1 引言

自上世纪90年代以来,因软件故障引发的重大事故层出不穷,如1996年欧洲阿丽亚纳5号火箭事件和1999年法国救护车事件等。软件故障已逐渐成为安全关键系统(safety-critical system)的主要故障源[1~3]。软件安全(safety)问题越来越突出,许多国家特别是西方发达国家正投入大量的人力、物力进行新安全保障技术的研究。究其原因在于两个方面:一是由于软件因其良好的可修改性和灵活性,常常被用来代替物理硬件,成为控制系统中各系统组件间、系统与环境间交互作用的控制和协调器,安全关键系统自身越来越复杂。其软件错误数,特别是设计型错误呈指数级增加。另一方面,从硬件借鉴的一些传统安全、可靠性分析法,如SFTA(软件故障树分析方法)和测试法,对于设计型软件错误而言已渐感力不从心[2]。据统计,就一般复杂度的软件,用测试法只能使软件错误率降低到10^{-4}个/h,对于复杂度稍高的软件,测试法还达不到这种效果。而安全关键系统的错误率要求是10^{-9}个/h,甚至有的是10^{-10}个/h[4]。如何走出这种困境?许多国家正在积极探讨。目前的研究角度分为两类:一是研究新的软件工程技术和新的安全、可靠性分析法,以尽量减少设计型软件错误,比如把硬件的FTA、FMEA和SCA扩展到软件领域。二是承认软件错误存在的前提下研究如何保障系统安全,包括全部和局部软件分隔技术。后者最具代表性的是Levenson等人提出的安全核(safety kernel)[5]。由于构思巧妙,它一直备受研究。Rushby利用形式化方法对安全核进行了描述[6],而Kevin则为两个模型系统MSS(Magnetic Stereotaxis System)和UVAR(University of Virginia Research Reactor)构建了相应的安全核,并在原型系统中得到了检验[7]。后来,Anderson等人把安全核改进为安全壳(safety shell)[8]。安全壳在原理上与安全核相似,只不过它把安全策略进一步细分为状态监视器、时间监视器和安全异常处理器三个子模块。另外,基于安全核(壳)的软件构架及与编程语言的关系也是研究热点,如基于反射机制Open C++和CML的实现技术[9]等。安全核与信息安全融合技术[10]目前也备受关注。

*)基金项目:福建省2003年青年科技人才创新基金(2003J020)、福建省2004年自然科学基金(A0410004)、云南省2003年自然科学基金(2003F0016M)、云南省信息技术专项(2004IT10)、厦门大学院士引进基金。**黎忠文** 博士,副教授,研究领域:实时系统高安全和高可靠技术;**姚绍文** 博士,教授,研究领域:分布式处理、Web技术、网络协议工程。

安全核自身的可信性这一关键问题又如何来解决呢？当前采用的方法是测试和限制安全核尺寸相结合的方法，这种方法极大地制约了安全核技术在复杂系统中的应用。为了突破这种限制，我们开展了多级安全核的研究[1]，它从限制安全核尺寸的角度，把大核分解为小核。这种方法尚待解决的难题是子核的划分和它们间的相互关系。基于此，本文提出了从安全需求分析开始到安全核生成过程中，如何通过形式化的方法来辅助提高安全核可信性的方法，为安全核技术在复杂系统中的应用提供了一种新思路。

2 安全核的可信性问题

与信息安全核(security kernel)不同的是，安全核关心的是如何保护设备，它将受保护设备与系统的其它部分隔离开，通过实施安全策略对这些设备进行特殊保护。凡是对受保护设备的访问，都必须经过安全核的审查控制。合法者予以支持，反之则采取相应的出错处理措施。这样就能很好地避免软件错误造成的灾难后果。

2.1 安全核的可信性

安全核的可信性主要体现在两个方面，图1是系统安全需求、安全策略和安全核的相互关系。其中，①根据系统的安全需求制定安全策略；②把①所制定的安全策略封装在安全核内；③和④分别表示对②和①的正确性进行验证，即④验证①结束后所制定的安全策略是否正确地反映了系统的安全需求，③同理。

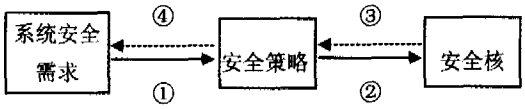

图1 系统安全需求、安全策略和安全核的关系

显然从表面上看，只须通过图1中的①和②就能建立安全核。然而，我们却很难断言这时的安全核能保障系统的安全，这是因为：图1中①完成后所制定的安全策略并不一定能通过④的验证，即安全策略不一定能满足系统的安全需求；同理，②结束后，由于过程②极有可能引入错误，因此封装在安全核内的安全策略也不一定就是①制定的安全策略，必须要通过③的验证。③和④步，目前主要采用测试技术。因此安全核的可信性取决于系统安全需求与安全策略的一致性、安全策略和安全核的一致性。

2.2 存在的问题

面对安全核的可信性问题，目前的方法是限制安全核尺寸加测试相结合的方法。尽量使安全核短小精干，便于测试是设计安全核的出发点[6]，文[6]给出了由安全核保障的安全策略必须满足的条件：

$$\forall a \in op^* \cdot P(a) \qquad (1)$$

其中 op^* 是安全核提供的所有功能组成的集合，其元素(安全以 a 代表，op 代表 a 所包含的具体功能)可以是安全核的一个或多个功能组成；$P(a)$ 则表示安全核对 a 包含的所有功能的输入/输出都有监控权。事实上，安全核为系统提供的每一次安全服务(实际上就是系统其它组件对设备的一次访问)，都可看成是一系列有序的安全核的功能集(即 a)。因此(1)式说明，无论系统其它组件(特别是应用软件)如何访问设备(即 op^*)，安全核都要保持安全策略的实施(即 $P(a)$)。

Kevin 在此基础上细分安全策略，提出了强安全策略的概念，进一步减少安全核的尺寸。然而系统的复杂化必将导致安全核尺寸的增加，以致其故障行为超过测试的掌控之中。因此安全核技术在复杂系统中的应用受到限制。

3 系统安全需求与安全策略的一致性

3.1 安全关键系统的抽象模型

一般来说，控制系统的工作过程是从传感器等一些监视装置中获取被控对象的信息，然后通过执行部件对被控对象进行控制操作。控制系统安全与否，就看被控对象是否会造成人生、财产损失或环境破坏。因此我们把控制应用软件及其支持环境看成控制司令部，于是控制系统的工作原理可抽象为图2所示。

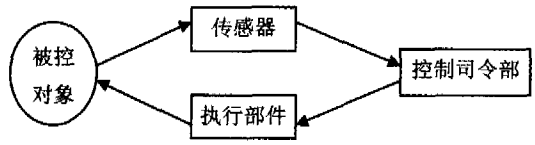

图2 控制系统的工作原理图

3.2 安全策略的制定工具

如何分析和制定真正能反映系统需求的安全策略是一个NP难问题。当前的研究集中在扩展硬件的可靠和安全性分析工具，如故障树分析法(FTA)、故障模式效应分析法(FMEA)和潜藏回路分析法(SCA)及其变种。这部分不是本文的研究重点。为便于说明，这里采用类故障树进行系统安全需求分析。为了确保安全需求分析结果的正确性，我们在安全需求分析的同时进行安全验证。安全验证包括两个方面：

1)安全需求分析树上，相邻层结点间安全的一致性验证。

2)同层结点间的非矛盾性(根据它们之间的与、或关系而定)的验证。

安全策略制定过程中必须注意：

1)没有通过安全验证的安全需求分析结果必须立即进行调整，不允许把当前层的问题带到下一层。

2)下层的安全需求分析工作从其上层的安全需求分析结果开始。

3)层与层之间存在反复，即当某层的安全需求分析无论如何都不能通过本层的安全验证时，工作就必须退回到其直接上层。以此类推。

3.3 一个实例——交通灯安全策略的制定

下面以十字路口交通灯为例，说明控制系统中安全策略制定的过程，而不致力于全面分析交通灯，因此在讨论中进行安全需求分析时做了一些假设。十字路口的交通规则是：

1)某方向上交通由通行变为停止的整个过程中，该方向上交通灯颜色相应的变化过程为：绿色→黄色→红色；

2)当交通由停止变为通行时，相应交通灯颜色的变化过程为：红色→红色闪→绿色。

为方便说明，此处不区分红色与红色闪，令：

1)十字路口 Cross 的方向 Direction::$=d_1 \mid d_2$ 且 $d_1 \rightarrow d_2 \in$ conflict；

2)灯色 light(d)和 light'(d)分别表示方向 d 上交通灯的当前和后继的颜色；

3)Timer 为计时器；

4)Lastchange(d)是十字路口 d 方向上交通灯最近一次

改变颜色时的时钟值；

5) yellowdelay 与 reddelay 分别表示交通灯必须保持黄色和红色的时间延时。

3.3.1 被控对象

3.3.1.1 安全需求分析

① 事故识别

仅以事故 DAC 为例，令：

事故 DAC 为 Cross 上来自 d_1，d_2 方向的车发生碰撞。

② 危险分析

假设导致 DAC 的危险 DHZ 只有一种：Cross 的两盏交通灯均为绿色，即：

$\forall\ d_1, d_2$: Direction | $d_1 \rightarrow d_2 \in$ conflict • light(d_1) $\cap$ (light(d_2) = green

③ 安全限制的制定

安全限制 DSC 为 d_1，d_2 两方向的交通灯不能同时为绿，即：

$\forall d_1, d_2$: Direction | $d_1 \rightarrow d_2 \in$ conflict • light(d_1) $\cap$ (light (d_2) $\neq$ green

④ 安全策略的制定

为了满足上述安全限制的要求，这里制定了三个安全策略 DSP1，DSP2 和 DSP3。

• 安全策略 DSP1

正常情况下，交通灯在变成红色之前，该灯一定保持了足够长时间的黄色，即：

$\forall d$: Direction | (light(d) = yellow) $\wedge$ (light$'$(d) = red)

• Timer-Lastchange(d) $\geqslant$ yellowdelay

• 安全策略 DSP2

正常情况下，某交通灯在变成绿色之前，处于冲突方向的交通灯一定保持了足够长时间的红色，即：

$\forall d_1, d_2$: Direction | $d_1 \rightarrow d_2 \in$ conflict $\wedge$ (light(d_1) = red) $\wedge$ (light$'$(d_1) = green)

• Timer-Lastchange(d_2) $\geqslant$ reddelay

• 安全策略 DSP3

两交通灯的物理功能是正确的。于是被控对象的安全需求分析树如图 3 所示。

3.3.1.2 安全验证

现在对上述安全需求分析进行安全验证，包括父子与兄弟两个方面。

① 父子之间一致性的验证

• 验证 DAC 到 DHZ 这一步的推导是否正确。

证明：

∵根据前面的假设，导致 DAC 的危险只有 DHZ 一种。

∴(¬DHZ⇒¬DAC) = TRUE 成立。

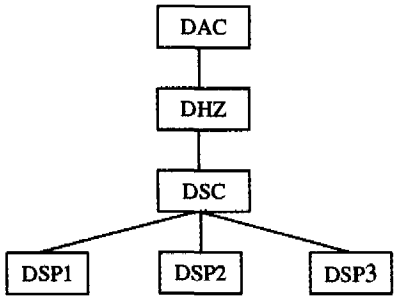

图 3 交通灯安全需求分析树

故 DAC 与 DHZ 之间的推导是正确的，从而就保证了它们之间安全需求的一致性。

• 验证 DSC 到 DHZ 这一步的推导是否正确

证明：

∵(¬DSC⇒DHZ) = TRUE

∴DSC 保证了 DHZ 不会发生，因此它们之间的推导是正确的，从而就保证了它们安全的一致性。

• DSP1，DSP2，DSP3 与 DSC 之间的关系

证明：

∵(¬DSP1 $\wedge$ ¬DSP2 $\wedge$ ¬DSP3⇒¬DSC) = FALSE

∴DSC 与 DSP1，DSP2，DSP3 之间的推导是不正确的，必须重新进行这一层的安全需求分析。

假设重新进行安全需求分析后所作的修改是加入了安全策略 DSP4。

安全策略 DSP4：正常情况下，至少存在一盏交通灯为红色，即：

$\forall d_1, d_2$: Direction | $d_1 \rightarrow d_2 \in$ conflict • light(d_1) $\cup$ light (d_2) = red

现在验证 DSP1，DSP2，DSP3，DSP4 与 DSC 之间的关系。

∵ (¬DSP1 $\wedge$ ¬DSP2 $\wedge$ ¬DSP3 $\wedge$ ¬DSP4 ⇒ ¬DSC) = TURE

∴DSC 与 DSP1，DSP2，DSP3，DSP4 之间的推导是正确的，从而就保证了它们之间安全的一致性。

② 兄弟之间非矛盾性验证

证明：

∵DSP1 $\wedge$ DSP2 $\wedge$ DSP3 $\wedge$ DSP4 = TRUE

∴DSP1，DSP2，DSP3，DSP4 之间是非矛盾的。

3.3.2 传感器与执行部件

传感器与执行部件的安全需求分析要接着交通灯的安全需求分析树的叶子结点进行。由于叶子结点中安全策略 DSP3 与传感器和执行部件无关，因此 DSP3 的实施由交通灯这一层自行处理，传感器与执行部件只考虑安全策略 DSP1，DSP2，DSP4。下面以传感器为例。

3.3.2.1 安全需求分析

① 事故识别

以破坏交通灯层的安全策略为事故，于是本层的三个事故分别用¬DSP1，¬DSP2 和¬DSP4 表示，下面以¬DSP4 为例进行分析。

② 危险分析

SHZ：表示传感器 S 把其所监视的灯的颜色不正确地传递给控制司令部；

OHZ：其它危险。

③ 安全限制的制定

对于危险 SHZ，令安全限制 SSC：传感器 S 必须把其所监视的交通灯的颜色按时且正确地传递给控制司令部；

假设传感器对于危险 OHZ 无能为力，因此把它作为需求分析树上的叶子结点，留给下层处理。

④ 安全策略

为了满足安全限制 SSC 的要求，这里制定了三个安全策略 SSP1，SSP2，SSP3，令它们为类似于硬件冗余、看门狗这样的措施。

传感器与执行部件层以事故¬DSP4 为根的安全需求分

析树如图 4 所示。

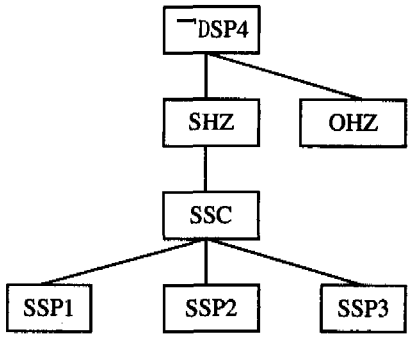

图 4 ¬DSP 的安全需求分析树

3.3.2.2 安全验证

安全验证类似于上一环节。

3.3.3 控制司令部

由于 SSP1,SSP2 和 SSP3 是硬件措施,它们与控制司令部这一层无关,因此这里只处理叶子结点 OHZ。在进行与上两层类似的安全需求分析和安全验证后,所制定的安全策略就是强安全策略,封装在安全核中的是与交通灯控制软件有关的强安全策略。

下面研究安全核与强安全策略的一致性问题。DSP1,DSP2 和 DSP4 是本例的安全策略。

4 安全核与安全策略的一致性

令 DSP1,DSP2 和 DSP4 具体封装在安全核内的抽象数据结构 SP 中,于是安全核与强安全策略的一致性就体现在 DSP1,DSP2 和 DSP4 与 SP 之间。我们探讨从软件需求说明的形式化表达到软件的形式化描述的问题,即根据 DSP1,DSP2 和 DSP4 进行 SP 的形式化描述。这样做的好处在于:一方面从形式化研究的发展状况来看,软件的形式化描述到软件的自动生成,是形式化研究要达到的最终目标之一,很多专家都在为之努力。因此 SP 的形式化描述有助于日后 SP 的自动生成(这时的 SP 才有望是无缺陷的)。另一方面,对 SP 形式化描述后,有助于从形式化的角度验证 SP 的安全性和活性。安全性是指绝不发生有害的事情,活性指某些好的事情会发生。这种验证可以进一步提高 SP 的可信性。

4.1 SP 的形式化描述

下面用 Z 语言描述 SP。

令:Cross 的工作模式 Mode 分为正确和故障两种,设

Mode::=right | error;

Color::=red | green | yellow;

reddelay, yellowdelay 分别为正确情况下交通灯应保持的红色和黄色的时延,它们是常数;

Tred(d)表示方向 d 上交通灯维持红色的时延;

Tyellow(d)表示方向 d 上交通灯维持黄色的时延;

Timer 计时器;

Lastchange(d)是十字路口 d 方向上交通灯最近一次改变颜色时的时钟值;

Clock 是时钟。

4.1.1 抽象数据结构 SP 的总体描述

SP 的总体描述如下所示,其中中间横线上是 SP 的变量,横线下是这些变量应满足的不变式:

SP

M:Mode
d:Direction
Tred(d)
Tyellow(d)
Lastchange(d)
Timer

$M=right \Rightarrow \forall d_1,d_2:Direction \cdot$
$(light(d_1) \cup light(d_2)=red) \wedge (\forall d: Direction \cdot$
$(Tred(d) \geqslant reddelay) \wedge (Tyellow(d) \geqslant yellowdelay(d))$

4.1.2 SP 至少应包括的操作

① SP 的初始化操作

令 SP 初始化操作为置灯 d_1 与 d_2 一盏为红色,另一盏为绿色;计时器 Timer 清零,并开始计时。SP 初始化操作的描述如下所示:

SPInit

同上

$M=right \Rightarrow \forall d_1,d_2:Direction \cdot (light'(d_1)$
$=red \wedge light'(d_2)=green) \vee ($
$light'(d_1)=green \wedge light'(d_1)=red)$
$Timer'::=Clock$

"'"表示初始化操作后相应变量的状态或值。

② 交通灯变成红色的操作 Cred

设 d 方向交通灯变成红色。Cred 的形式化描述如下所示:

Cred

d:Direction

---

light(d) =yellow
Timer-lastchange(d)≥yellowdelay
light'(d) =red
……

③ 交通灯变成绿色的操作 Cgreen

设方向 d 上的交通灯变成绿色,$d?$ 代表与 d 冲突的方向。Cgreen 的形式化描述如下所示:

Cgreen

d?:Direction

---

light(d) =red
$\forall d?:Direction \mid d \rightarrow d? \in conflict \cdot$
$light(d?)=red \wedge Time\text{-}lastchange(d?) \geqslant reddelay$
light'(d) =green

4.2 SP 与安全策略一致性的验证

现在验证上面描述的 SP 是否正确地封装了 DSP1、DSP2 和 DSP4。

①DSP4

它是 SP 总体描述中的不变式,显然 SP 封装了 DSP4。

②DSP2

DSP2 是由 SP 的操作 Cgreen 来实现的,现在证明之。

令 DSP2 中,d_2 是 $d?$,则

$\forall d_1,d?:Direction \mid d_1 \rightarrow d? \in conflict \wedge (light(d_1)=$

red) ∧ (light′(d_1)=green)

· Timer-Lastchange(d?)≥reddelay

显然 Timer-Lastchange(d?)≥reddelay 正是 Cgreen 所要维护的不变式。

③DSP1

显然 DSP1 是由 SP 的操作 Cred 维护的不变式。

因此,SP 是正确地封装了 DSP1、DSP2 和 DSP4。

5 原型实验

5.1 交通灯控制系统及其安全需求

实验室中模拟的交通灯控制系统 TCS 由两个十字路口(CROSS1 和 CROSS2)和一个交通指挥中心组成(见图 5)。其中每个十路口都有一个交通指挥系统,它通过传感器发来的交通流量情况改变两盏交通灯的颜色(比如在路口 CROSS1,水平方向的交通灯为 Light1,垂直方向的交通灯为 Light2)来指挥本路口的交通;交通指挥中心的任务是协调这两个路口的交通指挥系统,保证整个系统交通畅通。比如当图 5 中 CROSS1 路口箭头方向的车流量突然增大时,就需要交通指挥中心向 CROSS2 的交通指挥系统发出指令,要求其采取紧急措施,减缓 CROSS1 的压力。在交通灯正常工作的情况下,TCS 的安全需求是:交通指挥系统 1 和 2 无论收到指挥中心的指令与否都要正确地改变交通灯的颜色,避免同一个路口出现两个方向的交通灯同时为绿的情况。

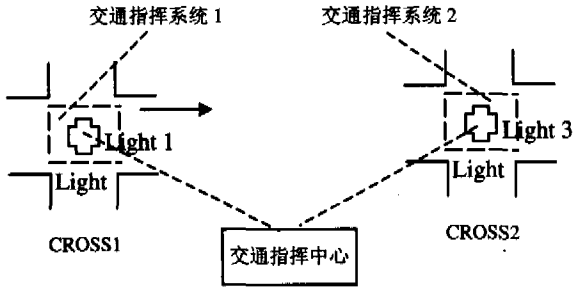

图 5 交通灯控制系统 TCS

实验室分别用三台 PC 机仿真 TCS,PC 机通过局域网与 HUB 相连。仿真 TCS 采用的软件是 RT-Linux,其版本为 Kernel 2.2.14-RT-Linux 2.2。其中,PC1 仿真交通指挥中心,PC2 和 PC3 分别仿真 CROSS1 和 CROSS2。PC2、PC3 各有一个安全核,其上的安全策略见上节。

5.2 安全核的实现

安全核的实现方式一般分为集中式和分散式两种,而且都是置于应用层的位置。我们设计把安全核放入实时操作系统(RTOS)中,安全核机制仍然由操作系统来提供,但不把它与操作系统的内核融为一体,而是把它作为一个单独的机制与操作系统的内核分开。这样,一方面能更多地考虑安全核与应用的关系;另一方面又能更好地体现灵活性,不需要安全服务时,系统就不提供。我们分析了 CRTOS Micro 和 RT-Linux 两个 RTOS。安全核机制的提供方式分为下列几种:

①库的方式:生成.o 文件,在编译时联入。

②扩展系统调用:相当于增加了用于安全的系统调用。

③修改系统函数 API:即在原 API 上加入类似于转向安全核的功能。

④内核模块法:这种方法用于实时 Linux。把安全核做成一个核心模块加入到实时 Linux 的内核中去。

我们这里采用的是第 4 种方法。

5.3 安全核对系统效率的影响

实验中每当安全核连续处理了 5 个有错的交通灯访问命令(实验中,概率为 0.2)时,它就与 PC1 通信一次。PC2 内采用安全机制前后,系统效率之比 η 为 98%,即采用安全机制后 PC2 的效率是原系统效率的 98%,用于安全的系统开销并不大,安全机制在本原型实验中是可行的。

总结 复杂分布式控制系统域内安全研究面临的困难之一是安全核的可信性问题,目前普遍认可的方法是测试。本文深入分析了安全核可信性的本质,并以交通灯控制为例,提出了从安全需求分析开始到安全核生成过程中,如何通过形式化的方法来提高安全核可信性的方法,为安全核技术在复杂系统中的应用提供了一种新思路。信息安全是当前的研究热点,然而法国救护车事件、美国 9.11 事件给人们敲响了警钟,控制系统的安全技术必将越来越受到人们的关注,只有 security 和 safety 技术的同步发展才会从根本上提高生活质量。

参 考 文 献

1 陈光宇,黄锡滋. 软件可靠性学科发展现状及展望[J]. 电子科技大学学报,2002,4(3):99~102

2 Zalewski J, Ehrenberger W, Saglietti F, et al. Safety of computer control systems: challenges and results in software development [J]. Annual Reviews in Control, 2003, 27:23~37

3 Leveson N G. System safety in computer—controlled automotive systems[R]. SAE Congress, 2000

4 Butler R W, Finelli G B. The infeasibility of quantifying the reliability of life—critical real—time software[J]. IEEE Tran on Software Engineering, 1993, 19(1):3~12

5 Leveson N G, Shimeall T J, et al. Design for Safe Software[A]. In: Proceedings AIAA Space Sciences Meeting[C], Reno, Nevada, 1983

6 Rushby J. Kernels for safety?. In: Safe And Secure Computing Systems Symposium. London: Blackwell Scientific Publications, 1989. 210~220

7 Kevin R. Safety kernel enforcement of software safety policies: USA: University of Virginia, 1995

8 Sahraoui A E, Anderson E, Katwijk V, et al. Formal specification of a safety shell in real-time control practice[A]. In: Proceedings of the WRTP'S 2000, 25thIFAC workshop on real-time programming[C], Oxford: Elsevier, 2000. 117~123

9 Sanz R, Zalewski J. Pattern-based control systems engineering [J]. IEEE Control Systems, 2003, 23(3): 43~60

10 黎忠文,熊光泽,李乐民. 分布式系统安全保障新体系的研究[J]. 电子学报,2003,31(4): 564~568

11 杨仁平,熊光泽,桑楠. 安全关键实时系统高可信集成技术的研究[J]. 电子学报,2003, 31(8):1237~1241