文章编号:1001 - 9081(2010)03 - 0708 - 07

基于不干扰理论的信道控制策略及其自动化验证方法

崔 隽[1,2],黄 皓[1,2]
(1. 南京大学 软件新技术国家重点实验室,南京 210093; 2. 南京大学 计算机科学与技术系,南京 210093)
(ctops@sina.com)

摘 要:通过研究信道与那些向其输入信息或从其获得信息的信息域之间直接或间接的干扰关系,来定义信道的语义和作用。明确描述和严格控制系统模块和进程之间的信息通道,有利于最大限度地保障模块或进程的完整性和可控性。所提出的信道控制策略正是基于上述目的。而针对信道控制策略复杂而不便于手工验证的特点,提出了基于通信顺序进程(CSP)的系统和策略描述方法以及基于 FDR2 的系统信息流策略自动化验证方法。该方法能够在少量的人工参与的情况下有效地分析信道控制策略,发现大部分存储隐蔽通道。

关键词:不干扰模型;信道控制;信息流;通信顺序进程;形式化验证
中图分类号: TP301 **文献标志码**:A

Channel control strategy based on noninterference theory and its automated verification scheme

CUI Jun[1,2], HUANG Hao[1,2]
(1. *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing Jiangsu* 210093, *China*;
2. *Department of Computer Science and Technology, Nanjing University, Nanjing Jiangsu* 210093, *China*)

Abstract: The authors defined the semantic description and functions of channels based on the study of the direct or indirect relation between any information domain and other domains who sent information to it or received from it. Exactly defining and strictly controlling the information channels between system modules or processes was beneficial to the integrity and controllability of the modules or processes. And in this paper, the new channel control strategy was used for this purpose. Channel control strategies were generally not easy to be manually verified because of their complexity. The authors presented an approach of describing system and strategies based on Communicating Sequential Processes (CSP) syntax and verifying the strategies in systems with automated verification tools FDR2. This approach can effectively and efficiently analyze the information channels and find out most of the storage covert channel.

Key words: noninterference model; channel control; information flow; Communicating Sequential Processes (CSP); formal verification

0 引言

进程或模块的隔离和信道控制是大多数软件系统或网络服务都可能采用甚至是不可或缺的安全防护措施。隔离有利于阻止系统的关键数据或代码受到恶意访问;有利于保护系统进程或模块的执行不受干扰;有利于阻断错误或失败在进程或模块之间蔓延,保证系统稳定性和可靠性。然而,隔离并不一定在所有场合都是绝对适用的。很多模块或者进程之间都必须交互和协作,但是系统的设计和维护者又往往希望这种交互是可控的、可追踪的,甚至在很多情况下希望证明某种交互方式是进程或模块间相互影响的唯一方式。比如,在微内核操作系统(如 minix)中,进程之间的通信只有消息这一种方式,这种方式是易于控制和追踪的。因此,信道控制越来越受到人们的关注,很多学者希望通过对信息流的研究来描述信道控制策略。不干扰理论是目前验证信息系统安全的主要方法。1982 年,Goguen 等人[1]提出了不干扰理论。不干扰理论通过观察信息域中信息值的改变来感知信息的流动,而不是简单依靠对读写操作的考查来确定信息流向,因此,具有很好地排除确定性系统中各种存储隐蔽信道问题的能力。在此基础上,Haigh 等人[2]提出了非传递的不干扰策略模型,如图 1,它强调信道控制,信息域 A 和 C 之间不能直接通信,必须通过安全信道 B 传递信息。根据文中对不干扰策略的描述,安全信道 B 同样是一个信息域,并且根据该模型描述,如果策略允许域 B 从信息域 A 接收信息,且同时允许 B 向域 C 传送信息,则认为信息域 B 是可以为 A 向 C 传递信息的;反之,如果策略不允许 B 作为 A 和 C 之间的信道,则策略一定不会同时允许 B 与 A 或 C 交互信息,即如图 2(b) 所示。本文认为,上述策略忽略了另一种普遍存在于大型软件系统中的情形。在大多数 C/S 或 B/S 架构的系统中,所有客户端都可以与服务系统通信,但是服务的设计者和客户往往都希望服务系统能够保证客户的隐私性和隔离性,不能将一个用户的服务数据泄露给另一个用户。显然,这样的服务系统就不是作为客户通信的安全信道出现的,它不能为客户传递信息,但是它又必须能够与客户交互。如图 2(a),A,C 能与 B 通信,但是 A,C 只能分别与 B 中的某些对象交换数据,而这些对象之间不发生信息交互。比如,银行存款服务中为每个客户维护的卡号密码等信息之间不会被互相复制。因此,传统的非传递不干扰策略无法描述此类公共模块或进程不允许作为通信信道的信息流策略

收稿日期:2009 - 09 - 29;**修回日期**:2009 - 11 - 09。 **基金项目**:国家 863 计划项目(2007AA01Z409);江苏省高技术项目(BE2008124)。
作者简介:崔隽(1981 -),男,江苏海安人,博士研究生,主要研究方向:操作系统形式化验证、安全模型; 黄皓(1957 -),男,江苏海门人,教授,博士生导师,主要研究方向:网络安全。

需求。同样,已有的策略模型,如 BLP、DTE 和 RBAC 也不能描述此类情况所需要的策略,即明确禁止信息域之间通过某些公共第三方传递数据的信息流策略。本文则以信息流安全研究中被普遍使用的信息流理论不干扰理论为基础,以 CSP 为描述和推理工具,分析信息域之间信道控制的基本理论和方法,并以此作为信息流策略制定和验证的基础。本文提出的新的信道控制策略模型,能够精确描述信息流动的特性,并通过策略明确禁止或允许某些公共的第三方信息域作为信息交互的信道。新的信道控制策略可以同时满足图 2(a)和图 2(b)所示的策略需求。

安全属性的自动化验证密切关系着系统安全分析和策略验证方案的可行性和实施效率。软件系统往往庞大而复杂,人工验证几乎是不可行的。为了从根本上解决验证问题,必须为每一类安全策略设计一种有效的形式化自动验证方案。针对不干扰策略,Roscoe 等人[3-4]基于确定性重新给出了不干扰属性的定义,这为成功应用 FDR2[5] 的进程确定性检测工具来自动化验证不干扰策略提供了条件,并根据此研究分别给出了验证传递不干扰和非传递不干扰理论的方案[6]。本文仍沿用该思想,并根据信道控制理论的特点,设计了适合信道控制策略验证的解决方案,并通过具体的文件服务实例演示了系统描述和验证的过程。

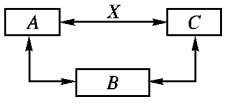

图 1　B 作为 A 和 C 之间的唯一信息通道

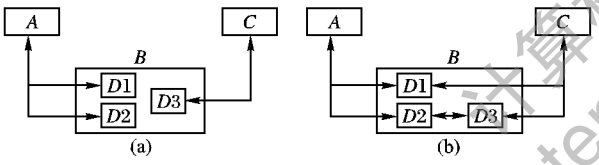

图 2　B 既可以隔离 A、C,也可以作为 A、C 的通信信道

1　非传递不干扰理论

Rushby[7] 定义了一个系统的有限状态自动机如下。

定义 1　系统 M 包括以下元素。

1) 系统状态集合 S,系统的初始状态记为 s_0。

2) 系统动作集合 A。

3) 系统输出集合 O。

4) 单步状态转换函数 $step: S \times A \to S$。描述了执行动作 A 前后的状态转换。

5) 系统输出函数 $output: S \times A \to O$。描述了执行动作 A 后的系统输出。

6) 系统信息域集合 D。对于任意的信息域 $u, v \in D$,有 $u \cup v \in D$。

7) 系统执行函数 $run: S \times A^* \to S$。描述了执行一系列动作后的状态转变。

8) 行为执行域函数 $dom: A \to D$,$dom(a)$ 表示执行行为 a 的信息域。

基于自动机,Rushby 定义了 $D \times D$ 上的不干扰关系 $\rightsquigarrow$ 及其补关系 $\not\rightsquigarrow = D \times D \setminus \rightsquigarrow$。系统的不干扰策略可以表达为系统中不同信息域之间的不干扰关系。为了能够验证一个系统设计是否满足给定的不干扰策略,Rushby 给出了满足给定不干扰策略的系统 M 的定义如下。

定义 2　给定系统 M,以及安全策略 $\rightsquigarrow$,如果 $\forall a \in A$, $\beta \in A^*$ 满足 $output(run(s_0, \beta), a) = output(run(s_0, ipurge(\beta, dom(a))), a)$,则称系统 M 满足安全策略 $\rightsquigarrow$(其中提取函数 $ipurge$ 的定义参见文献[7])。

由于定义 2 中对系统 M 满足安全策略的判定依赖于对提取函数 $ipurge$ 的计算,而 $ipurge$ 的计算又需要作用于任意的执行序列 β,即使序列是有穷的,人工判定该条件也是非常低效和不现实的。因此,Rushby[7]26-30 提出了等价于该判定条件的单步展开条件,虽然单步条件避免了对 $ipurge$ 的计算,易于判定,但仍需人工地对每一个系统的执行动作逐一判断,无法适用于大型的复杂系统验证过程。

为此,Roscoe 等人[6] 在定义 2 的基础上,基于 CSP 规范[8-9] 重新给出了系统 M 满足策略 $\rightsquigarrow$ 的条件。在该定义中,所有信息域都被描述为并发执行的进程,信息流动被描述为进程通信,而系统 M 则是由所有的这些通信进程构成的复合进程。于是,基于CSP规范给出的满足不干扰策略的系统 M 的定义如下:

定义 3　给定确定性系统 M 以及安全策略 $\rightsquigarrow$,如果 $\forall a \in A, u \in D, s, s' \in traces(M)$,满足 $s \setminus \alpha(noflow(u)) = s' \setminus \alpha(noflow(u)) \Rightarrow head(M/s) \cap \alpha(u) = head(M/s') \cap \alpha(u)$,则称系统 M 满足安全策略 $\rightsquigarrow$(其中,$noflow(u) = \{v \mid v \in D, v \not\rightsquigarrow u\}$,$Head(M/s)$ 为 M 在执行 s 后可能执行的动作集合)。

$head(M/s) \cap \alpha(u)$ 表达了在系统执行轨迹 s 之后立即可以执行的进程 u 上的动作集合。在 CSP 中,即使相同的输入或输出动作,如果输入或输出值不同则会被视为不同的动作。所以定义 3 中 $head(M/s) \cap \alpha(u)$,不仅能够包含定义 2 中 $output()$ 所表达的进程 u 的输出变化,还可以表达进程 u 在动作执行上发生的改变。定义 3 指出所有 $noflow(u)$ 中进程的执行都不会影响 u 上的输出和下一动作的执行。基于 CSP 给出的该条件可以借助成熟的形式化验证工具来验证,比定义 2 更容易判定系统 M 是否满足给定的不干扰策略。

2　基于不干扰理论的信道控制策略

2.1　信道控制策略的定义和描述

信道可以看作是一个信息域,信息流入或流出信道则也可以看作是信道与其他信息域之间的一种干扰关系。但是与传统的非传递的不干扰关系的语义不同,并不表示只要某个信息域 A 能够接受某个域 B 的信息,同时能够发送信息给另一个域 C,即 $B \rightsquigarrow A, A \rightsquigarrow C$,则域 A 就能够成为 B 和 C 之间的信息通道。相反,A 可能就是个信息分流控制器,B 发送来的信息只能传递给 C 以外的信息域。此类情况同样出现在 C/S 结构的系统中,很多服务进程不会充当两个客户进程的信息通道,而是独立地为每一个客户服务。此类策略是传统不干扰策略所不能描述的,本文将基于不干扰策略给出信道的不干扰语义,以及更为精细的信道控制策略描述。为了描述信道,定义 $D \times D \times D$ 上的新的干扰关系 $\rightarrow$ 及其补关系 $\not\rightarrow = D \times D \times D \setminus \rightarrow$。对于任意信息域 A、B、C,存在策略允许 A 作为 B 向 C 传递信息的信道,则该策略可记作 $B \overset{A}{\rightarrow} C$,反之记作 $B \overset{A}{\not\rightarrow} C$。根据不干扰理论和信道的定义显然有,如果信息域 A、B、C 满足 $B \overset{A}{\rightarrow} C$,则 $B \rightsquigarrow A, A \rightsquigarrow C$,但反之并不一定成立。且如果 $B \overset{B}{\not\rightarrow} C$ 或者 $B \overset{C}{\not\rightarrow} C$,则有 $B \rightsquigarrow C$。如果 $B \rightsquigarrow C$,其实也就不存在控制 B、C 之间的信道了,所以下文的定义和描述中均不考虑这种情况。

另一方面,动作的执行存在着继承关系,并且与执行前的状态相关。因此,还需为系统 M 增加判断任意状态下任意动作是否可执行的描述。于是在定义1中增加动作或动作序列的可执行判断函数。

定义4 系统 M 除包含定义1所有元素外还包括。

1) 动作可执行判定函数。$enabled: S \times A \rightarrow \text{Boolean}$, $enabled(s,a)$ 表示在状态 S 下动作 a 是否可执行。

2) 序列可执行判定函数。$enableds: S \times A^* \rightarrow \text{Boolean}$, $enableds(s,\gamma)$ 表示在状态 S 下序列 γ 是否可执行。

如果所有允许为信息域 u、v 传递信息的信道都未受到 u 的数据输入和执行的影响,则 u 的任意操作一定不会影响 v 的输出视图。基于此分析,类似定义2,给出满足信道控制策略的系统 M 的定义如下。

定义5 给定系统 M,以及不干扰策略 $\rightsquigarrow$,信道控制策略 PL,且对于任意 $u,v \in D, \alpha,\beta,\alpha',\beta' \in A^*$,满足:

$$u \not\rightsquigarrow v \wedge enableds(s_0,\alpha\circ\alpha') \wedge enableds(s_0,\beta\circ\beta') \wedge purge1(\alpha,u,v) = purge1(\beta,u,v) \wedge purge2(\alpha',u,v) = purge2(\beta',u,v) \Rightarrow outputs(\alpha,\alpha',u,v) = outputs(\beta,\beta',u,v)$$

则称系统 M 满足信道控制策略 PL。

其中输出集合:

$$outputs(\alpha,\alpha',u,v) = \{output(run(s_0,\alpha\circ\gamma'),u) \mid \gamma' \in A^*, enableds(s_0,\alpha\circ\gamma'), purge2(\alpha',u,v) = purge2(\gamma',u,v)\}$$

提取函数 $purge1$, $purge2$ 的定义如下:

$$purge1: A^* \times D \times D \rightarrow A^*$$

$$purge1(\Lambda,u,v) = \Lambda$$

$$purge1(a\circ\alpha,u,v) = \begin{cases} a\circ purge1(\alpha,u,v), & dom(a) \neq u \\ purge1(\alpha,u,v), & \text{其他} \end{cases}$$

$$purge2: A^* \times D \times D \rightarrow A^*$$

$$purge2(\Lambda,u,v) = \Lambda$$

$$purge2(a\circ\alpha,u,v) = \begin{cases} a\circ purge2(\alpha,u,v), & u \xrightarrow{dom(a)} v \text{ 或 } dom(a) = v \\ purge2(\alpha,u,v), & \text{其他} \end{cases}$$

定义5表达了以下内容:如果信息域 u 的执行不影响策略 PL 允许的 u,v 间的传递信道,则 u 的任意操作同样不会影响 v 的输出视图。$outputs(\alpha,\alpha',u,v)$ 描述的就是所有可能的输出结果集合,$outputs$ 集合中 v 上不同的输出结果仅与 u、v 之间信道以外的信息域及 u 有关。$purge1$ 仅去除了任意执行轨迹中的 u 的操作;$purge2$ 则不仅去除了 u 的操作,还去除了所有不允许作为信道在 u,v 之间传递数据的信息域上的操作。$u \not\rightsquigarrow v$ 说明了两个提取函数中的 $dom(a)$ 不等于 u、v 中的任一个。

与定义3类似,为了利用进程代数来自动化验证信道策略,下文将给出定义5的CSP描述如下。

定义6 给定系统 M,不干扰策略 $\rightsquigarrow$,信道控制策略 PL,如果对于任意 $u,v \in D, s_1,s_2 \in traces(M)$, $s_3 \in traces((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))$,满足

$$u \not\rightsquigarrow v \wedge s_1\backslash\alpha(u) = s_2\backslash\alpha(u) \wedge s_3 \in traces((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u)) \Rightarrow head(((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v) = head(((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v)$$

则称系统 M 满足信道控制策略 PL,其中,$unchl(u,v) = \{w \mid u \overset{w}{\not\rightsquigarrow} v\}$。

$unchl(u,v)$ 定义了不能为 u、v 之间传递信息的域的集合。输出集合 $outputs$ 的相等条件的判断通过比较集合中元素是否相同来实现,其中 v 上可能的输出结果类似于定义3用 $head(M/s_1) \cap \alpha(v)$ 来描述。定义6有利于得到基于进程代数的判定定理,从而使用进程代数相关的推理和自动化验证方法和工具来判定系统 M 是否满足给定的信道控制策略,发现违背策略的隐通道或设计错误。

根据定义6,如果能够对于系统中任意两个不同的执行路径 s_1, s_2,按照定义中的判定条件进行判断就可以断定任意两个信息域之间是否存在策略允许的信道以外的信息交互通道。但是,很显然这个工作不能是人工的,用CSP重新定义不干扰条件的目的就是要利用一些形式化验证工具(如FDR2)去自动验证。然而,即使是由软件自动化验证,数量庞大的验证分支仍可能使系统验证的过程低效甚至是无法在可接受的时间范围内完成。为了提高验证的效率,将对 s_1, s_2 尽可能地作一些限制。

首先,在一个实际系统中,把任意信息域都看作进程,把进程间的同步消息机制作为进程间的唯一通信方式。对于系统中原有的异步通信或共享内存的通信方式可以通过构建一个或多个第三方进程来暂存异步消息或操作共享内存数据。而这些第三方进程同样按照同步通信的方式与原系统进程交互,在基于CSP的描述方法中正是利用这种思想来描述一个实际的复杂系统的。于是,就可以将任意进程 u 上的动作划分为两类:同步通信动作集合 $sig(u)$ 和内部动作集合 $inner(u)$。通信动作一定是成对出现的,并且是由通信的发起方和接收方同时分别执行自己的通信动作来实现信息传递或同步,为描述方便,可认为通信接收的动作总是紧接在通信发起动作之后执行的。而内部动作因为无需和其他进程同步,它的执行对其他进程是透明的,也不可能对其他进程有任何直接的影响,即使有任何影响,也要通过通信动作间接地将干扰传递出去。

其次,对于任意信息域 u、v 之间的信道的考查,我们关心的就是 u 的不同操作会通过怎样的信道反映在 v 的输出或操作上,$s_1\backslash\alpha(u) = s_2\backslash\alpha(u)$ 也说明了 u 是唯一的干扰源。因此,从直观上看,无需考查所有的 s_1, s_2,而只需关心以不同的 u 的通信动作结尾的 s_1, s_2 执行轨迹对。即满足 $s_1\backslash inner(u) = s_2\backslash inner(u)$, $tail(s_1) \neq tail(s_2)$ ($tail(t)$ 表示 t 执行的最后一个动作)的序列 s_1, s_2。下面将非形式地说明这一点。

首先,如果存在序列 s_1,s_2,s_3 如定义6所描述,且 $tail(s_1) = tail(s_2)$,则根据前面的分析,通信事件一定是成对出现的,而 $s_1\backslash\alpha(u) = s_2\backslash\alpha(u)$,所以 s_1, s_2 中或者不存在 u 上的通信动作,或者存在着相同的通信动作。如果 s_1,s_2,s_3 不满足 $head(((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v) = head(((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v)$。则不妨设存在 v 上的一个操作 $c \in \alpha(v)$,有 $<c> \in head(((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v)$, $<c> \notin head(((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v)$。根据前面的分析,$u$ 的内部事件不能直接影响 v 的输出和执行,所以 s_1, s_2 在 u 上的差异不会直接导致 v 上动作 c 能否执行,而是通过 u 上不同的通信动作在 s_1, s_2 执行结束之后,s_3 执行过程中去直接影响 v 中动作 c 的执行或者通过 $unchl(u,v)$ 间接影响 v。即一定存在轨迹 t_1, t_2 分别紧接着 s_1, s_2 之后执行,有 $t_1\backslash inner(u) = t_2\backslash inner(u)$, $tail(s_1) \neq tail(s_2)$,且 t_1, t_2 在 v 和 u、v 信道上执行的轨迹是 s_3 的一部分,即存在 s_4,有 $s_3 = (t_1\backslash\alpha(u)\backslash\alpha(unchl(u,v)))^\frown s_4$。所以,根据上述讨

论一定有 $<c> \in head(((M/(s_1\hat{}t_1))\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_4) \cap \alpha(v)$，$<c> \notin head(((M/(s_2\hat{}t_2))\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_4) \cap \alpha(v)$，即 $head(((M/(s_1\hat{}t_1))\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_4) \cap \alpha(v) \neq head(((M/(s_2\hat{}t_2))\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_4) \cap \alpha(v)$。

以上讨论说明了对于定义 6 中定义的序列 s_1,s_2，如果最后一个动作不满足 $tail(s_1) \neq tail(s_2)$，则依据定义 6 的判定条件验证的结果一定可以归结为判定另外一对满足 $tail(s_1') \neq tail(s_2')$ 的执行序列 s_1',s_2'，且 s_1',s_2' 中至少有一个是以 u 上通信动作结尾的。所以定义 6 的验证范围可以被缩小，并重新定义如下。

定义 7　给定系统 M 以及不干扰策略 $\rightsquigarrow$，信道控制策略 PL，如果对于任意 $u,v \in D, s_1,s_2 \in traces(M), s_3 \in traces((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))$，满足

$$u \rightsquigarrow v \wedge s_1\backslash inner(u) = s_2\backslash inner(u) \wedge tail(s_1) \neq tail(s_2) \wedge s_3 \in traces((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u)) \Rightarrow head(((M/s_1)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v) = head(((M/s_2)\backslash\alpha(unchl(u,v))\backslash\alpha(u))/s_3) \cap \alpha(v)$$

则称系统 M 满足信道控制策略 PL。

定义 7 使得只需验证当 s_1,s_2 首次执行 u 上不同的通信动作，且以它们结尾的情况，其他情况的验证结果可以由上述情况推出。

2.2　信道控制策略的自动化验证方法

定义 7 已经采用 CSP 的描述形式，但是并不能直接交由形式化验证工具 FDR2 来验证，因为它还不是 FDR2 可验证的断言形式，还需根据 FDR2 可验证断言的要求，构造可证明满足定义 7 的可验证断言。

FDR2 可验证断言包括进程确定性、死锁、活锁，进程间提炼和等价等断言类型，本方案将使用进程提炼检查来验证。由 2.1 节可知，CSP 将信息域描述为进程，则定义相关进程和集合如下。

定义 8　给定确定性系统进程 M 以及不干扰策略 $\rightsquigarrow$，信道控制策略 PL，对于任意 $U,V \in D$，有如下定义。

1) 进程 $A = \cup\{w \mid w \in D, U \overset{w}{\nrightarrow} V\}, B = \cup\{w \mid w \in D, U \overset{w}{\rightarrow} V\}$，显然 $A,B \in D$。

2) 进程 A'、B'、U'、V' 是进程 A、B、U、V 的镜像，$A',B',U',V' \in D$；且 $\forall a \in \alpha(A)$，有唯一的镜像操作 $a' \in \alpha(A')$，B'、U'、V' 也有类似的性质，且 A'、B'、U'、V'、A、B、U、V 八个进程的字母表也没有交集。由 A'、B'、U'、V' 类似于 A、B、U、V 的通信方式构成的系统进程记为 M'。

3) $tag(U'),tag(A'),tag(V')$ 是 U'、A'、V' 的字母表的一个拷贝。且 $\forall a \in \alpha(U')$ 有唯一的 $ta \in tag(U')$，$tag(U')$ 与 $\alpha(U')$ 无交集。$tag(A'),tag(V')$ 也有类似的性质。

4) 不确定进程 $L(M) = M\backslash inner(U)[\alpha(U)]CHAOS(U)$，其镜像 $L(M') = M'\backslash inner(U')[\alpha(U')]CHAOS(U')$。

5) 同步 M 和 M' 的通信进程：$Com1,Com2$。

$$\begin{aligned}
Com1 &= (?x:\alpha(U) \rightarrow tx \rightarrow ((x' \rightarrow Com1)\square(?y':\alpha(U') - \{x'\} \rightarrow ty' \rightarrow Com1')\square Com1')\square \\
&\quad (?x:\alpha(A) \cup \alpha(V) \rightarrow x' \rightarrow Com1)\square \\
&\quad (?x:\alpha(B) \rightarrow tx \rightarrow x' \rightarrow Com1) \\
Com1' &= (?x:\alpha(U) \cup \alpha(B) \rightarrow ((?y':\alpha(U') \cup \alpha(B') \rightarrow Com1')\square Com1'))\square(?x:\alpha(A) \cup \alpha(V) \rightarrow ((x' \rightarrow tx' \rightarrow Com2)\square \\
&\quad (?y':\alpha(U') \cup \alpha(B') \rightarrow Com2''(x)))) \\
Com1''(x) &= (x' \rightarrow tx' \rightarrow Com1')\square(?y':\alpha(U') \cup \alpha(B') \rightarrow Com1''(x)) \\
Com2 &= (?x:\alpha(U) \rightarrow tx \rightarrow ((x' \rightarrow Com1)\square(?y':\alpha(U') - \{x'\} \rightarrow ty' \rightarrow Com2')\square Com2')\square \\
&\quad (?x:\alpha(A) \cup \alpha(V) \rightarrow x' \rightarrow Com2)\square \\
&\quad (?x:\alpha(B) \rightarrow tx \rightarrow x' \rightarrow Com2) \\
Com2' &= (?x:\alpha(U) \cup \alpha(B) \rightarrow ((?y':\alpha(U') \cup \alpha(B') \rightarrow Com2')\square Com2'))\square(?x:\alpha(A) \cup \alpha(V) \rightarrow ((x' \rightarrow tx' \rightarrow Com2)\square \\
&\quad (?y':\alpha(U') \cup \alpha(B') \rightarrow Com2''(x))\square \\
&\quad (tx' \rightarrow STOP))) \\
Com2''(x) &= (x' \rightarrow tx' \rightarrow Com2')\square(?y':\alpha(U') \cup \alpha(B') \rightarrow Com2''(x))
\end{aligned}$$

6) 复合通信进程：$R1(M,U,V)$。

$$\begin{aligned}
R1(M,U,V) &= L(M)[\alpha(M)]Com1[\alpha(M')]L(M')\backslash\alpha(U)\backslash\alpha(U')\backslash\alpha(A')\backslash\alpha(B)\backslash\alpha(B')\backslash tag(A')\backslash\alpha(V') \\
R2(M,U,V) &= L(M)[\alpha(M)]Com2[\alpha(M')]L(M')\backslash\alpha(U)\backslash\alpha(U')\backslash\alpha(A')\backslash\alpha(B)\backslash\alpha(B')\backslash tag(A')\backslash\alpha(V')
\end{aligned}$$

由定义 8 不难看出，$Com1,Com2$ 定义 $L(M)$ 与 $L(M')$ 之间的同步规则，$Com1,Com2$ 始终要求 $L(M')$ 同步执行 $L(M)$ 在域 U 以外的操作的镜像，直到 $L(M')$ 在 U' 上首次未同步执行 $L(M)$ 在域 U 上执行的操作的镜像。在 $L(M)$ 与 $L(M')$ 出现了首次分歧之后，$Com1,Com2$ 也转向执行 $Com1',Com2'$。$Com1',Com2'$ 使得 $L(M')$ 仍保持同步执行 $L(M)$ 在域 A 和 V 上的操作的镜像，但不再要求 $L(M')$ 与 $L(M)$ 在域 B 上的操作同步，允许域 B,B' 和 U,U' 上的操作自由执行。很显然，如果域 U 不干扰 V，且 U 的操作不能通过影响 B 而间接影响 V 的话，只要 $L(M)$ 与 $L(M')$ 在 A 和 A' 上执行同步的序列，则 V 和 V' 上可以执行的序列集合也应该是一致的，即 V 上的可执行序列的镜像也一定可以在 V' 中执行。$Com1'$ 与 $Com2'$ 正是通过同步 V 和 V' 的操作来匹配 V 的可执行序列和其在 V' 上的序列镜像，其差别仅在于，当 $Com1'$ 或 $Com2'$ 与 $L(M)$ 同步执行了 $\alpha(A)$ 或 $\alpha(V)$ 中的动作 x 后，如果 $L(M')$ 的某个执行分支当前无法执行相应的镜像动作 x' 时，会因为无法与 $Com1'$ 或 $Com2'$ 同步而阻塞，而 $Com2'$ 在阻塞之前，会抢先执行了 tx'，而 $Com1'$ 则会直接阻塞该分支。即当 $L(M')$ 在域 A' 或 V' 上无法执行与 $L(M)$ 同步的镜像操作时，假设 $L(M)$ 执行的是 c 操作，则 $R1$ 中会出现 $c \rightarrow STOP$ 的执行序列，而 $R2$ 中会出现 $c \rightarrow tc' \rightarrow STOP$ 的执行序列。然而，如果 $L(M')$ 在域 A' 或 V' 上可以执行与 $L(M)$ 同步的镜像操作时，则 $R1$ 和 $R2$ 中均会出现 $c \rightarrow tc'$ 的执行序列。显然，这可以用于判断 $L(M')$ 和 $L(M)$ 是否可以同步执行某个域 A 或域 V 上的操作序列。在下文的分析中将会指出 $R1$ 与 $R2$ 的上述差别，将有助于判断域 U 是否仅通过域 A 而非 B 传递信息给 V。判定定理如下：

定理 1　给定确定性系统 M 以及不干扰策略 $\rightsquigarrow$，信道控制策略 PL，对于任意 $U,V \in D$，如果 $R1(M,U,V),R2(M,U,V)$ 符合定义 8，则系统 M 满足信道控制策略 PL 的充要条件是：$traces(R2(M,U,V)) \subseteq traces(R1(M,U,V))$。

证明　即证明 $traces(R2(M,U,V)) \subseteq traces(R1(M,U,V))$ 与定义 7 的条件等价。

1) 充分性。

用反证法：证明如果对于任意的执行序列 $s_1,s_2 \in traces(M)$，$s_1\backslash\alpha(U) = s_2\backslash\alpha(U) \wedge tail(s_1) \neq tail(s_2)$，$s_3 \in traces((M/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))$，满足定义7的前提条件，但是不满足 $head(((M/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) = head(((M/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，则 $traces(R2(M,U,V)) \subseteq traces(R1(M,U,V))$ 不成立。

首先，根据 $Com1$ 和 $L(M)$ 定义可知，$L(M)$ 和 $L(M')$ 第一次没有在 U 和 U' 上同步执行的操作一定是通信动作，所以可以假设从初始状态到 $L(M)$ 和 $L(M')$ 第一次执行不分歧的通信动作为止 $L(M)$ 和 $L(M')$ 分别执行了 s_1 和 s_2 的镜像 s_2'，显然，s_1,s_2 满足 $s_1\backslash\alpha(U) = s_2\backslash\alpha(U) \wedge tail(s_1) \neq tail(s_2)$。又 $L(M)$ 和 $L(M')$ 隐藏了 U 和 U' 的内部事件，所以，一定存在 $a_1,a_2 \in \alpha(U)$，$s_{12} \in L(M)$，s_{12} 的镜像 $s_{12}' \in L(M')$，$s_1 = s_{12}^\frown <a_1>$，$s_2 = s_{12}'^\frown <a_2>$。根据 $R1$ 的定义可知，$R1$ 隐藏了 U、U'、A'、B、B'、V' 的操作，仅保留 B,V 的操作，但是 $Com1$ 使得 s_{12},s_{12}' 中的 U、B 的操作被 $tag(U)$、$tag(B)$ 的标记所替代，而 a_1,a_2 被 ta_1,ta_2 所替代，于是存在 sp 为 s_{12} 中 U、B 的操作用 $tag(U)$、$tag(B)$ 的标记替代后的轨迹，则有 $sp^\frown <ta_1>^\frown <ta_2> \in traces(R1)$。再根据 $Com1'$ 和 $Com1''$ 的定义可知，$Com1'$ 和 $Com1''$ 始终要求 $L(M')$ 和 $L(M)$ 在 A、V 及其镜像域上保持同步，而在 U、B 及其镜像域上 $L(M)$ 和 $L(M')$ 可以选择任意的动作执行。于是，在 $L(M)$ 和 $L(M')$ 分别执行了 s_1，s_2' 之后，到某个任意状态为止，如果 $L(M)$ 和 $L(M')$ 能够在 A、V、A'、V' 上同步，设执行的轨迹记作 t_1,t_2'，则显然有 $s_3 = t_1\backslash inner(U)\backslash\alpha(unchl(U,V))$ 以及 s_3 的镜像 $s_3' = t_2'\backslash inner(U')\backslash\alpha(unchl(U',V'))$。根据 $R1$ 和 $Com1'$ 的定义可知，$R1$ 屏蔽了 U、U'、A'、B、B'、V' 的操作，仅保留 B,V 的操作，同时 $Com1'$ 使得 t_2' 中的 V' 的操作被 $tag(V')$ 的标记所替代，即存在轨迹 tp，其等价于 s_3 中屏蔽 U、B 的操作，并在 V 的任意操作之后添加 $tag(V')$ 的标记而构成的轨迹。根据 $R1$ 的定义可知，$sp^\frown tp \in traces(R1)$。如果 $head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) \neq head(((L(M')/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，则一定存在 V 的某操作 c，有 $<c> \in head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，$<c'> \notin head(((L(M')/s_2')\backslash\alpha(unchl(U',V'))\backslash\alpha(U'))/s_3') \cap \alpha(V')$，则根据 $R1$ 和 Com' 的定义知 $sp^\frown tp^\frown <c>^\frown <tc'> \notin traces(R1)$。相反，若 $head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) = head(((L(M')/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，即对于任意 V 上操作 c，如果有 $<c> \in head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，则有 $<c'> \in head(((L(M')/s_2')\backslash\alpha(unchl(U',V'))\backslash\alpha(U'))/s_3') \cap \alpha(V')$，即 $sp^\frown tp^\frown <c>^\frown <tc'> \in traces(R1)$。同理，根据 $R2$ 和 $Com2$、$Com2'$、$Com2''$ 的定义，同样有 $sp^\frown tp \in traces(R2)$。但是根据 $Com2'$ 可知，$Com2'$ 与 $Com1'$ 的区别仅在于当 $L(M')$ 在域 A'、V' 上无法与 $Com2'$ 同步时，$Com2'$ 会在阻塞前执行阻塞事件的标记操作，即如果阻塞事件是 c'，tc' 将会被执行，而 $Com2'$ 中 c' 一定出现在 c 之后，所以一定有 $sp^\frown tp^\frown <c>^\frown <tc'> \in traces(R2)$。这与上文中 t_1，t_2' 在 U、B、U'、B' 上的操作选择无关。根据上述分析可以看出，如果对于任意的 s_1,s_2,s_3，有 $head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) \neq head(((L(M')/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，则一定存在某个 V 上的操作 c，有 $sp^\frown tp^\frown <c>^\frown <tc'> \notin traces(R1)$，而 $sp^\frown tp^\frown <c>^\frown <tc'> \in traces(R2)$，则显然有 $traces(R2(M,U,V)) \subseteq traces(R1(M,U,V))$ 不成立。

2) 必要性。

用反证法：证明如果存在 $ts^\frown <e> \in traces(R1(M,U,V))$，$ts^\frown <e> \notin traces(R2(M,U,V))$，则一定存在执行序列 $s_1,s_2 \in traces(M)$，$s_1\backslash\alpha(U) = s_2\backslash\alpha(U) \wedge tail(s_1) \neq tail(s_2)$，$s_3 \in traces((M/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))$，满足定义7的前提条件，但是不满足 $head(((M/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) = head(((M/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$。

根据 $R1$ 和 $R2$ 的定义可知 $R1$ 与 $R2$ 的区别仅在于 $Com2'$ 在无法与 $L(M')$ 在 A'，V' 上同步时，执行被阻塞操作（如 c'）的标记事件 tc'，而 $Com1'$ 则直接阻塞该操作。因此，在出现上述不同步之前，$R1$ 和 $R2$ 可执行的轨迹是相同的。即如果存在假设中的情况，则 e 一定存在于 $\alpha(A')$、$\alpha(V')$ 或 $tag(A')$、$tag(V')$ 中，而 $R1$、$R2$ 都屏蔽了 U、U'、A'、B、B'、V'、$tag(A')$ 的操作，则一定有 $e \in tag(V')$，即一定是 $Com2'$ 在无法与 $L(M')$ 在 V' 上同步时产生的。而在这种情况下，根据1) 中的分析知，一定存在 s_1,s_2,s_3 及其镜像 s_1',s_2',s_3'，有 $c \in head(((L(M)/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$，$c' \notin head(((L(M')/s_2')\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3') \cap \alpha(V)$，其中 $e = tc'$。即 $head(((M/s_1)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V) = head(((M/s_2)\backslash\alpha(unchl(U,V))\backslash\alpha(U))/s_3) \cap \alpha(V)$ 不成立。

综合1)、2) 所述，命题得证。

3 基于文件系统的实例验证

根据定理1，信道控制策略的验证可以通过构造提炼断言来实现。信道控制策略的验证方案可以应用于大多数系统设计中，用以验证系统设计中是否存在着与设计目标相悖的隐通道。以一个增加了并发机制的 minix 文件系统设计为例，通过模拟多个文件系统线程并发执行，可以分析线程之间相互影响的信息通道。两个线程之间不一定完全隔离，不同的线程之间可以受控地通过全局对象传递信息。因此，具有并发执行能力的文件系统设计应该能够满足：线程对共享资源的访问是受控的。分析和验证文件系统中是否存在上述不受控制的线程间信息流动，即存储隐通道，就是下文将要阐述的。在给出文件系统的信息流策略之前，先给出文件系统的一个抽象描述。

为了简化描述，定义文件仅包含四个基本操作：*Open*，*Read*，*Write*，*Close*，和五类全局内存对象：打开文件列表 *fl*，描述一个进程打开了哪些文件，打开文件链表的元素指向打开文件的文件描述符；文件描述符表 *filp*，描述了系统所有打开文件的文件使用情况，每一个文件描述符都关联一个 *inode* 节点；文件 *i* 节点表 *inode*，描述了文件本身的相关属性，与文件载入的磁盘块关联；磁盘块表 *block*，存储文件载入的内容；锁描述表 *lock*，为实现上述四类对象的互斥访问而设置。另外，为了简单起见，假设该文件系统处理的对象只包括文件。于是，基于 CSP 描述规范，可以描述支持并发机制的文件服务如下：

$FS = Open[\,]Read[\,]Write[\,]Close; FSs =$

$[\,]i:\{1\cdots maxthread\}@FS$

FS 为一个文件服务的线程,而 *FSs* 是多个线程的并发描述,以 *Open* 操作为例,其描述如下所示:

```
Open = openRequest?pid?filename -> isopend!pid!filename ->
  replyopenedfl?fli ->
if fli > -1 then replyReqO.pid!fli -> FS
else getfreefl!pid -> replyfreefl?fli -> getfreefilp!pid?fd ->
  if fli > -1 and fd > -1 then lock!pid!filplock!fd ->
    locked?bOK1 ->
    if bOK1 then isfdused!fd -> replyfdused?bOK3 ->
      if bOK3 then unlock!filplock!fd -> replyReqO.pid!(-1) -> FS
      else isexist!filename -> replyexistino?ino ->
        if ino > -1 then lock!pid!inodelock!ino -> locked?bOK2 ->
          if bOK2 then setfilp!pid!fli!fd ->
            setinode!fd!ino -> addicount!ino -> unlock!inodelock!
            ino -> unlock!filplock!fd -> replyReqO.pid!fli -> FS
          else unlock!filplock!fd -> replyReqO.pid!(-1) -> FS
        else getfreeinode!pid?ino ->
        if ino > -1 then lock!pid!inodelock!ino -> locked?bOK2 ->
          if bOK2 then isinoused!ino -> replyinoused?bOK3 ->
            if bOK3 then unlock!filplock!fd ->
              unlock!inodelock!ino -> replyReqO.pid!(-1) -> FS
            else setfilp!pid!fli!fd -> setinode!fd!ino ->
              setinocount!ino!1 -> initinode!ino!filename ->
              unlock!inodelock!ino -> unlock!filplock!fd ->
              replyReqO.pid!fli -> FS
          else unlock!filplock!fd -> replyReqO.pid!(-1) -> FS
        else unlock!filplock!fd -> replyReqO.pid!(-1) -> FS
    else replyReqO.pid!(-1) -> FS
  else replyReqO.pid!(-1) -> FS
```

Open 操作首先接收用户请求 *openRequest*,如果请求的文件已经在该进程的打开文件列表中,则返回打开文件的文件描述符;否则为该进程的打开文件列表和文件描述符列表搜索一个空闲的表节点,并为它们增加互斥锁,用于今后存放新打开的文件。如果该文件已经被其他用户进程打开,则该文件的 *inode* 节点已经载入内存,搜索该节点,并通过 *setfilp* 和 *setinode* 操作与之前找到的打开文件列表和文件描述符表空闲节点建立联系,并为它们解锁。如果该文件未被系统中任何进程打开过,则需要在 *inode* 表中搜索一个空闲节点,并从磁盘上载入该文件节点,并如上建立与打开文件列表和文件描述符表中节点的联系。最后如果成功返回打开文件描述符节点的序号,否则返回 -1。*Write*、*Read*、*Close* 操作也有类似的描述。

由上文中对 *Open* 操作的描述可知,描述中并没有实现具体的共享对象操作,而是通过输出事件(如:*setfilp*、*setinode* 等)请求数据操作,而具体实施的对共享对象的操作用专门的数据服务进程来描述。定义文件系统对上述五类对象的操作进程依次为:$D1(fl)$、$D2(filp)$、$D3(inode)$、$D4(block)$ 和 $D5(locks)$。以 $D1(fl)$ 为例描述如下所示:

```
D1(fl) = setfilp?pid?fli?fd -> D1(setfd(fl,pid,fli,fd))
  [] getfilp?pid?fli -> replyfilp!getfd(fl,pid,fli) -> D1(fl)
  [] getfreefl?pid -> replyfreefl!getfirstfreefl(fl,pid) -> D1(fl)
  [] isopend?pid?filename -> isexist!filename -> replyexistino?ino ->
  if ino > -1 then isexistfilp!ino -> replyexistfilp?fd ->
    if fd > -1 then replyopenedfl!checkopen(fl,pid,fd,filename)
      -> D1(fl)
    else replyopenedfl!(-1) -> D1(fl)
  else replyopenedfl!(-1) -> D1(fl)
```

$D1(fl)$ 描述了四个对文件打开列表的操作,即关联文件描述符 *setfilp*,取得文件描述符 *getfilp*,取得空闲的打开文件列表节点 *getfreefl*,查询文件打开状态 *isopend*。$D1(fl)$ 可以作为独立于文件操作和其他对象操作的进程看待,同时由于 $D1(fl)$ 是对全局对象 *fl* 的读取操作,因此也为所有文件系统的线程所共享,不同线程都可能通过 $D1(fl)$ 传递信息。同理,$D2(filp)$,$D3(inode)$,$D4(block)$,$D5(locks)$ 也同样可能作为线程之间的信息传递通道。而除此之外,线程之间不再存在其他的共享内存对象。于是,可以描述文件系统线程共享数据服务如下:

$$FSSystem = ((((((FSs[|\,aD1\,|]D1(fl))[|\,aD2\,|]D2(filp))[|\,aD3\,|]D3(inode))[|\,aD4\,|]D4(locks))[|\,aD5\,|]D5(block))$$

其中:*aD1*,*aD2*,*aD3*,*aD4*,*aD5* 是 *FS* 向数据服务进程请求的操作集合,如 $aD1 = \{setfilp, getfilp, getfreefl, isopend\}$。其他数据服务进程的描述与其类似。

虽然五类共享对象的操作进程都可能是线程间相互影响的传递信道,但是并不一定任意两个线程都需要共享这五类对象中的元素。比如,如果两个文件服务线程处理的是来自不同进程的请求,它们就不会共享进程打开文件列表对象 *fl* 和文件描述符 *filp*;如果它们也不共享文件,则 *inode* 节点也不会共享。从分权和最小特权的角度考虑,任何线程只应该访问它完成文件服务所需要的资源。具体地说,根据线程服务的用户进程之间的资源共享关系,存在下述安全需求。

1) 请求文件服务的多个进程之间不存在文件的共享,为该类进程服务的文件服务线程则不会共享上述五类全局对象中的任意对象元素。

2) 请求文件服务的多个进程之间存在文件的共享,但不是父子进程关系,为该类进程服务的文件服务线程则不会共享进程打开文件列表对象 *fl* 和文件描述符 *filp* 对象元素。

3) 请求文件服务的是同一个进程的多个线程或者是父子进程,为该类线程服务的文件服务线程则允许共享上述五类全局对象中的任意对象元素。

上述三种情况中所说的无共享是相对于请求文件服务的用户进程或线程而言的,文件服务线程可能在争抢同一类全局对象的节点中感知其他线程的存在,但是服务请求进程不关心文件服务进程使用什么节点为其服务,只要能得到正确的服务结果即可。所以,在不考虑上述五类节点资源耗尽的情况下,请求服务的用户进程是不应该从上文提到的非共享对象上感知其他用户进程的存在的,则认为为这些用户进程服务的线程也是不共享这些对象的。上述第 1) 种情况说明两线程之间无通信信道,第 2) 种情况认为部分信道是不可用的,第 3) 种情况认为所有信道都可用。下文将以第 2) 种情况为例,制定第 2 章提到的信道控制策略,并根据上述验证方案,验证第 2) 种情况的设计正确性。

如果将不同的服务线程和五类对象的数据操作进程都看作独立的信息域:*FS*1、*FS*2、$D1(fl)$、$D2(filp)$、$D3(inode)$、$D4(block)$、$D5(lock)$,则根据第 2 章对信道控制策略的定义,以及上文中对第 2) 种情况的描述,可以得到下述信道控制策略:

$$FS1 \nrightarrow FS2,\ FS1 \overset{D1(fl)}{\nrightarrow} FS2,\ FS1 \overset{D1(filp)}{\nrightarrow} FS2,$$
$$FS1 \overset{D1(inodel)}{\rightarrow} FS2,\ FS1 \overset{D1(blockl)}{\rightarrow} FS2,\ FS1 \overset{D1(fl)}{\rightarrow} FS2$$

于是,根据上文对文件系统的 CSP 描述,可以计算定义 8

中的对象如下：

1) $M = ((((((FS1[\ ]FS2)[|\ aD1\ |]D1(fl))\ [|\ aD2|]D2(filp))[|\ aD3\ |]D3(inode))[|\ aD4\ |]D4(locks))[|\ aD5\ |]D5(block))$；

2) $U = FS1; V = FS2; A = D3(inode) \cup D4(locks) \cup D5(block); B = D1(fl) \cup D2(filp)$；

3) $R1(M,U,V) = L(M)[\alpha(M)]Com1[\alpha(M')]L(M') \setminus \alpha(U') \setminus \alpha(A') \setminus \alpha(B')$。

将 M 及 $FS1$、$FS2$、$D1(fl)$、$D2(filp)$、$D3(inode)$、$D4(block)$、$D5(lock)$、$R1$、$COM1$、$Spec$ 的 CSP 描述输入 FDR2，执行验证断言 $AssertSpec[F = R1]$，即可验证上述信道控制策略，结果如图 3 所示。

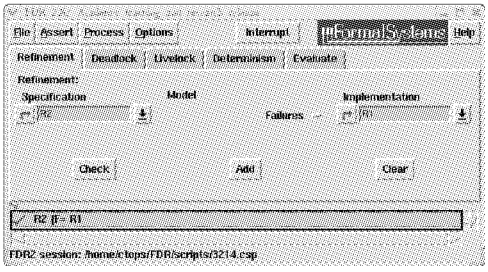

图 3 FDR2 信道控制策略验证结果

显然，上述方案可以应用在复杂信息系统的信息流策略分析中。本文所定义的信道控制策略刻画了各个模块之间安全的信息交互的设计需求，而 2.2 节给出的验证方案则可以用于验证系统设计能否满足根据上述安全需求所定义的信道控制策略。

4 结语

本文基于不干扰理论分析了系统中信息域之间的直接或间接的交互关系。而用于在两个不直接交互信息域之间传递数据的中间域就是信道。本文通过研究信道与那些向其输入信息或从其获得信息的信息域之间直接或间接的干扰关系来定义和描述信道。信息普遍存在于复杂系统中的任两个功能模块或进程之间，明确描述和严格控制系统模块和进程之间的信息通道，有利于最大限度地保障模块或进程的完整性和可控性。本文提出的信道控制策略正是用于基于上述目的信道描述。针对信道控制策略复杂而不便于手工验证的特点，本文提出了基于 CSP 的系统和策略描述以及基于 FDR2 的策略自动化验证方法。该方法能够在少量人工参与的情况下有效地分析信道控制策略，发现大部分存储隐蔽通道。

参考文献：

[1] GOGUEN J, MESEGUER J. Security policies and security models [C]// Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy. Los Alamitos: IEEE Computer Society, 1982: 11 - 20.

[2] HAIGH J, YOUNG W. Extending the non-interference model of MLS for SAT [C]// Proceedings of the 1986 Symposium on Security and Privacy. Oakland, CA: IEEE Computer Society, 1986: 232 - 239.

[3] ROSCOE A W, WOODCOCK J C P, WULF L. Non-interference through determinism [J]. Journal of Computer Security, 1996, 4 (1): 27 - 54.

[4] ROSCOE A W. CSP and determinism in security modeling [C]// Proceedings of the 1995 IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Society, 1995: 114 - 127.

[5] Formal Systems (Europe) Ltd. FDR 2 user manual [EB/OL]. [2009 - 07 - 20]. http://www.fsel.com/documentation/fdr2/html/index.html.

[6] ROSCOE A W, GOLDSMITH M H. What is intransitive noninterference? [C]// Proceedings of the 12th Computer Security Foundations Workshop. Mordano, Italy: IEEE Computer Society, 1999: 228 - 238.

[7] RUSHBY J. Noninterference, transitivity, and channel-control security policies [R]. Menlo Park: Stanford Research Institute, 1992.

[8] HOARE C A R. Communicating sequential processes [J]. Communications of the ACM, 1978, 21(8): 666 - 677.

[9] ROSCOE A W. The theory and practice of concurrency [M]. Upper Saddle River, NJ: Prentice-Hall, 1997.

(上接第 701 页)

5 结语

本文提出了一种基于聚类和时间序列模型的入侵检测算法，可以有效地将正常的数据和攻击数据区分开来，且具有很好的准确性。利用 KDD Cup 99 数据集的实验表明，这一算法在具有较高的检测率的同时保持了相对较低的误警率，特别是对于拒绝服务攻击更是如此。

参考文献：

[1] 杨智君，田地，马骏骁，等. 入侵检测技术研究综述[J]. 计算机工程与设计，2006，27(12)：2119 - 2123.

[2] PORTNOY L, ESKIN E, STOLFO S J. Intrusion detection with unlabeled data using clustering [C]// DMSA 2001: Proceedings of 2001 ACM CSS Workshop on Data Mining Applied to Security. Phila delphia, PA: ACM Press, 2001: 5 - 8.

[3] LAZAREVIC A, ERTOZ L, KUMAR V, *et al*. A comparative study of anomaly detection schemes in network intrusion detection [C]// Proceedings of the 3rd SIAM Conference on Data Mining. New York: ACM Press, 2003: 801 - 813.

[4] 向继，高能，荆继武. 聚类算法在网络入侵检测中的应用[J]. 计算机工程，2003，29(16)：48 - 185.

[5] 罗敏，王丽娜，张焕国. 基于无监督聚类的入侵检测方法[J]. 电子学报，2003，31(11)：1713 - 1716.

[6] 李卫平. k-means 聚类算法研究[J]. 中西部科技，2008，7(8)：52 - 53.

[7] 赵铁山，李增智，高波，等. 时间序列模型在入侵检测中的应用研究[J]. 计算机工程与设计，2005，26(5)：1128 - 1129.

[8] 陈铁梅，黄道平，陆顾新，等. 模式聚类在数据预处理中的应用研究[J]. 计算机与应用化学，2003，20(3)：241 - 243.

[9] University of California. KDD Cup 1999 DATASETS [EB/OL]. [2009 - 04 - 20]. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[10] LEE W. A data mining framework for building intrusion models [C]// Proceedings of the 1999 IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Society, 1999: 120 - 132.

[11] HAN JIAWEI, KAMBER M. 数据挖掘：概念与技术[M]. 北京：机械工业出版社，2003.

[12] TENG SHAO-HUA, ZHANG WEI, ZHU ZHO-HUI, *et al*. DDoS attack detection and defense based on feature and data fusion [J]. System and Information Sciences Notes, 2007, 1(4): 390 - 395.