多核平台上针对 seL4 的分区机制研究

丁贵强[1] 王雷[2*] 王鹿鸣[3] 康乔[4]
（北京航空航天大学计算机学院 北京市 100191）[1,2,3]
（北京航空航天大学软件学院 北京市 100191）[4]

摘 要 在航空电子等嵌入式领域中，多核时代已经来临，如何充分利用多核成为了现在系统领域的研究热点。同时由于系统集成度越来越高，一个硬件平台可能需要同时运行不同安全级别的任务，这就需要操作系统为应用提供隔离与保护。为了解决这两个问题，本文在目前只支持单核的 seL4 基础上分别加入多核和分区隔离的支持，之后又提出多核和分区机制结合的方案，实现带分区机制的多核 seL4，最终运行在 qemu 模拟器上，分区机制的实现符合 ARINC653 标准的语义。

关键词 seL4，多核，分区，嵌入式

中图法分类号 TP311 **文献标识码** A DOI

The study of partition mechanism about seL4 on the multi-core platform

DING Gui-qiang[1] WANG Lei[2*] WANG Lu-ming[3] KANG Qiao[4]
(School of Computer Science & Engineering，Beihang University，Beijing 100191， China)[1,2,3]
(School of Software，Beihang University，Beijing 100191， China)[4]

Abstract In the field of avionics and other embedded, the multi-core era has come, how to make full use of multiple cores has become the research field of the current system. At the same time as a result of the system integration is getting higher and higher, a hardware platform may need to run different security levels of the task, which requires the operating system for the application to provide isolation and protection. In order to solve the two problems mentioned earlier, this paper only supports the single core seL4 based on the addition of multi-core and partition isolation support, and then put forward multi-core seL4 and partition mechanism to achieve a partition mechanism with multi-core seL4 run on the qemu simulator, and the partition mechanism is implemented in accordance with the semantics of the ARINC653 standard.

Keywords Sel4，Multi-core，Partition，Embedded

1 引言

seL4(Secure Embedded L4)是澳大利亚信息通信技术中心（NICTA）开发的一款微内核操作系统。本文在单核 seL4 的基础上加入多核支持，并引入符合 ARINC653 标准的分区隔离，并设计多核分区系统。论文组织如下：第二节讲述 seL4 多核系统设计，第三节介绍了分区的标准，主要是 ARINC653 标准然后在 seL4 中加入分区隔离的支持，第四节提出了多核设计与分区机制结合的方案，最后总结了本文完成的工作。

2 seL4 的多核设计

seL4 是第一个通过形式化方法被证明正确的微内核操作系统[1]。其在安全性、可靠性上有非常大的优势，但 seL4 目前只支持单核，这限制了 seL4 内核的应用范围。目前多数设备都采用了多核处理器。所以，实现多核支持对于扩展 seL4 的应用范围来说是十分必要的。

多核的引入需要考虑数据竞争和多核调度器

到稿日期：2017-7-15 返修日期：2017-9-10
本文受国家自然科学基金（No.61672073 和 61272167）资助。
丁贵强（1991-），男，硕士研究生，学生，主要研究方向为微内核操作系统、嵌入式，E-mail: 920398694@qq.com。王雷（1969-），男，副教授，通信作者，主要研究方向为操作系统，E-mail: wanglei@buaa.edu.cn。

两方面的问题。数据竞争问题是指在多核处理器下，内核的数据结构被各个核心共享。当内核在多个核心上同时运行时，内核的数据结构会发生数据竞争，即各个核心均会访问或修改同一个数据结构，从而造成错误。在解决数据竞争方面，普遍采用的方案是使用锁，比如 Linux、BSD 等均采用了细粒度的锁来保护内核的各类数据结构。而多核调度方面，将在 2.3 节讨论。

2.1 多核支持

在多种多核实现方案中，大内核锁方案实现简单，正确性易于得到保证，是比较适合本文的方案。

方案的总体设计图如图 1 所示。内核中的数据结构采用大内核锁进行保护。每个核心都设有单独的内核栈、处理器相关的私有状态以及当前运行的线程的指针。调度器继承单核下的调度器：全局共享唯一的一个调度队列，每次将队列中优先级最高的线程从队列中弹出，并设置为当前核心上正在运行的线程。

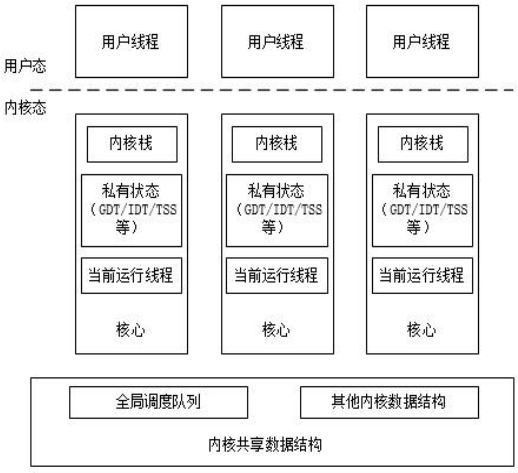

图 1 总体设计图

该方案的思路是将内核整个作为一个临界区，同一时刻只允许一个核心执行内核代码，从而避免数据竞争的问题。它的主要优点在于对代码的改动较小，只需在进入和退出内核时添加锁的获取和释放语句即可。seL4 由于系统调用的执行流程较短，大内核锁对于性能的影响是有限的。在贴近真实应用场景的性能测试中，大内核锁方案所表现出来的可扩展性与更细粒度的锁、RTM(Restricted Transactional Memory)锁等相近。因此，本方案对于 seL4 微内核来说性能是可以接受的[2]。

2.2 多核启动

为了启用处理器上所有的核心，需要依据 x86 的多核启动流程，实现适用于多核的系统启动代码。在 x86 多核处理器下，处理器被划分为 BSP（Bootstrap Processor）和 AP（Application Processor）两种。系统中存在一个 BSP，该处理器是系统启动时，第 1 个被启动的处理器，其余的处理器都被称作 AP。在系统启动时，BIOS 会在各个处理器上执行必要的初始化代码，之后将除了 BSP 以外的处理器全部置于 halt 状态，并将 BSP 移交给操作系统的初始化代码。

操作系统的初始化代码需要初始化当前处理器，建立操作系统所需的所有基本的数据结构，最后唤醒所有的 AP，在各个 AP 上执行对 AP 的初始化。整个流程如图 2 所示。

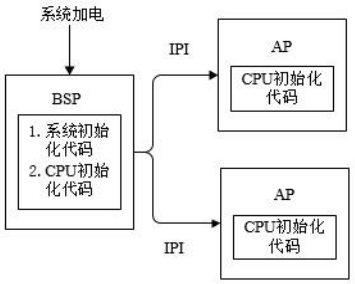

图 2 x86 多核启动流程

在 seL4 中实现多核启动需要解决三个问题：一是如何唤醒其它核心？二是如何设置各个核心运行时所需的内核栈？三是针对当前 AP 都需要初始化哪些数据？在多核处理器中，存在一个特殊的中断，名为核间中断，简称 IPI（Inter-Processor Interrupts）。IPI 是由一个核心发送给另一个核心的中断。通过向其它的 AP 发送 IPI 中断，操作系统的初始化代码就可以唤醒相应的 AP。IPI 分为很多种，用于初始化的 IPI 可以在发送中断时指定一个地址，收到中断的 AP 会自动跳转到该地址执行代码。在实现 seL4 多

核启动时，需要将 AP 的初始化代码拷贝到 AP 可以读取的一个地址上，之后向各个 AP 发出初始化 IPI，各 AP 就会跳转到 AP 初始化代码的入口处，开始执行 AP 的初始化工作。需要在 AP 上初始化的数据结构只和当前核心相关，每个核心只需要单独调用 init_cpu 函数即可完成初始化。

2.3 多核调度

seL4 原有的调度器是静态优先级调度。每次发生时钟中断时，调度器取出队首的线程，设置为当前正在执行的线程。之后退出内核空间时，内核代码会自动恢复当前正在执行的线程的上下文，从而实现线程的调度。

多核调度需要考虑的一个问题是如何在各个核心上实现调度。换句话说就是如何让别的核心执行代码，这一点只能通过中断来实现。通过产生中断，使得收到中断的核心进入到中断处理程序，进而在返回用户空间时，返回到被调度的线程的上下文中。有两种中断适用这一应用场景：核心自身的时钟中断以及核间中断。通过核间中断虽然可以使调度器在别的核心上执行代码。但由于大内核锁的存在，收到核间中断的核心一时难以进入到内核态，这影响了调度效率。因此，采用时钟中断来实现调度，就成为了首选。每当核心产生时钟中断时，进入到多核调度器代码中，调度器为当前核心调度一个线程。由于每个核心都会产生时钟中断，所以每个核心都会执行多核调度器的代码，从而实现为各个核心调度线程。

3 seL4 的分区机制

随着技术的发展，嵌入式硬件的性能得到大幅度提高，人们期望在同一硬件平台上集成多个应用程序，从而起到提高集成度，节约成本的作用，这就意味着需要将多种不同安全级别的程序集成在一起，为了解决该问题，人们提出由嵌入式操作系统通过分区的手段为应用提供隔离。目前分区的标准主要有两种，一是 MILS(Multiple Independent Levels of Security/Safety)[3]，二是 ARINC653(Avionics Application Standard Software Interface)标准[4,5]。其中 ARINC653 是应用较为广泛的分区操作系统标准。本文研究 seL4 支持 ARINC653 标准的可行性，设计一套基于 seL4 实现分区隔离的方案，帮助 seL4 能够真正应用于航空电子、医疗器械等高安全、高可靠性需求的嵌入式领域。

3.1 ARINC653 标准介绍

在 ARINC653 标准中，系统资源的使用单位、系统调度的单位有两种，一种叫分区（Partition），一种叫进程（Process）。其中分区是进程的容器，一个分区至少包含一个进程，而一个进程必定属于一个分区，分区与进程的关系类似于 UNIX 系统中进程与线程的关系。ARINC653 标准对分区、进程的概念、属性和状态进行了详细定义，并给出了调度、内存、通信等若干方面的要求。下面对这几方面进行简要的介绍。

3.1.1 分区管理

分区是 ARINC653 标准的核心体现。分区操作系统的不同功能模块应该被“隔离”，这里的隔离包括两个方面:空间上的隔离（内存隔离）和时间上的隔离（时间隔离）。分区存在的主要意义是为了容错，即一个分区的失效不能传播到其他分区中去。ARINC653 分区的主要属性如下表：

表 1 分区的主要属性

属性名称	英文名称	简介
分区标识符	partition id	唯一标识符
周期	period	分区周期性运行
持续时间	duration	每个周期内分区运行多长时间
分区模式	mode	分区的状态
内存限制	memory limits	分区使用内存上限

空间隔离指的是分区不能访问其他分区的内存。同时每个分区的内存大小是固定的，不能在运行时额外申请内存，比如一般不提供像 malloc 这样的系统调用；并且，分区内进程所使用的栈空间是固定的。所有进程的内存大小就是分区的内存大小。为了实现内存隔离，ARINC653 推荐采用 MMU 和页表相结合的方法，给每个分区分配一个地址空间，从而实现分区之间的隔离。由于不同的分区的页表不一样，在操

作系统的虚拟内存机制和 MMU 的保护下，一个分区自然就不能访问其他分区的内存。同时在创建进程的过程中，我们会指定每个进程的栈段大小。关于时间隔离，ARINC653 制定了一种独特的二级调度标准，如图 3 所示。只有时间落在分区对应的窗口内，对应的分区才处于活跃态，这个分区内的进程才能运行。宏观来看，调度具有周期性，一个周期叫做主时间窗。因此，分区调度器的功能就是支持这一循环往复的周期性调度。此外，分区的数量和每个分区的运行时间由用户决定（可配置），也就是说，图 3 所示的时间配置在系统启动后就不会再发生改变了。

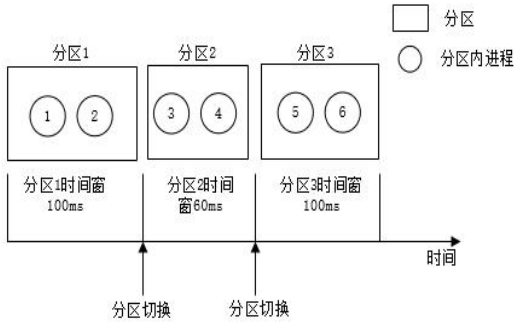

图 3　分区调度

3.1.2　进程管理

ARINC653 系统中，一个分区内有多个进程，但 ARINC653 并没有明确规定进程之间是否需要进行内存的隔离。因此，很多系统实际上都使用传统意义上的“线程”来模拟 ARINC653 分区中的进程，比如基于 Linux 的 ARINC653 操作系统[6]。这样一来，一个分区内的进程就完全共享分区的内存空间。分区内的进程具有的主要属性如表 2 所示：

表 2　进程的主要属性

属性名称	英文名称	简介
进程号	process id	唯一标识一个进程
周期	period	只对周期性进程有意义
优先级	priority	进程的优先级
状态	state	进程的状态

分区内的进程按照优先级调度，即优先级高的进程先执行。调度是可抢占的，如果出现了优先级更高的进程则会抢占当前的进程。实际上，分区内的调度算法可以由用户自定义。有的系统，比如 pok[7]都为分区内的进程调度提供了多种算法实现，用户可以根据需要选择合适的分区内调度算法，甚至是自行编写调度算法。

3.1.3　分区间通信

ARINC653 为分区之间提供基于端口（port）的通信方式，消息的发出和接收都要指定相应的端口。每个分区可以具有多个端口，端口要么是用来发送的，要么是用来接收的。一个分区的发送端口和另一个分区的接收端口就构成了一个单工的通信信道（channel），ARINC653 支持一对多的消息发送，即一个发送端口可以有多个目的端口。但不是所有的发送、接收端口都能够构成信道，所有的信道在系统构建时由配置文件指定。这样，只有具备信道的分区之间才有能力进行通信。端口实际上是保存在内核中的数据结构，当操作系统接收到分区通过端口发送消息的请求的时候，查找是否有端口与之构成信道。如果有，则把消息内容拷贝到目的端口中，否则就丢弃这个消息。操作系统不关心消息的具体内容，只是简单的进行消息的拷贝。

3.2　基于 seL4 的分区管理方法

3.2.1　分区在 seL4 中的表示方法

借鉴 Linux653 的做法，可以用 seL4 的进程表示分区，而用 seL4 的线程表示分区内的进程。从而利用 seL4 的 VSpace(Virtual Space)实现内存的隔离。这一映射方法如图 4 所示。

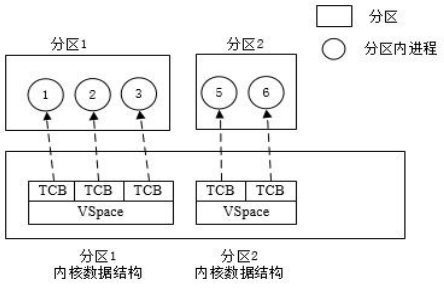

图 4　分区与进程在 seL4 中的表示

3.2.2　用户态分区管理方法

本文的分区方案是在用户态实现对分区的管理[8]，即在内核态与用户态之间，新加入一层，称为“分区管理层”，实际上，这个层就是一个进程（即 seL4 的根任务）。除了根任务的主线

程，还需要额外创建一个调度线程。图 5 中根任务与分区的图示是一样的，而调度线程与分区内进程的图示是一样的。

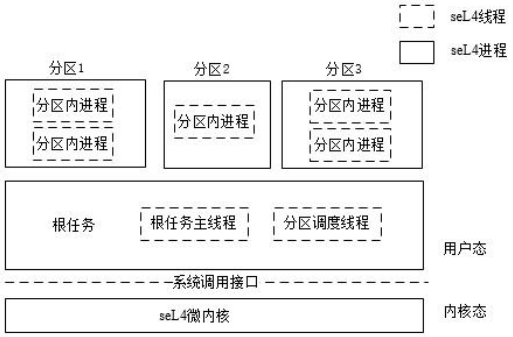

图 5　用户态分区管理层

seL4 内核启动之后，首先运行的就是根任务。根任务与其他分区实际上是平等的：他们在 seL4 内核的角度下都是用户态进程。根任务启动后的主要职责就是读取分区的配置信息，建立管理分区和分区内进程的数据结构，并调用 utils 库提供的接口请求内核创建 seL4 进程和 seL4 线程。为了允许其他任务与根任务进行通信，根任务需要创建一个同步端点（Endpoint），这样做的目的是为了提供 APEX(APplication EXecutive)接口的服务。当用户进程调用 APEX 接口的时候，实际上通过 seL4 的 IPC 给根任务主线程发消息来完成。由于发消息需要使用同步端点，因此根任务必须在初始化时创建这个端点。完成分区和进程的创建之后，根任务创建一个线程，即分区调度线程。由于分区调度线程与根任务共享地址空间，因此调度线程可以直接访问根任务建立的所有相关数据结构。创建完调度线程后，根任务就可以在端点上挂起，等待消息的到来。

3.2.3　分区内存隔离的保证

分区内存隔离一方面意味着不同分区之间不能互相访问，另一方面也意味着分区的内存总量是固定的。针对第一方面，由于用 seL4 的进程表示分区，因此 seL4 内核的 VSpace 实际上实现了分区之间的内存是不能互相访问的。当发生分区切换时，seL4 内核会执行页表的切换。针对第二个方面，由于分区内进程的数据段和代码段的长度在用户给定配置之后，就是确定的，并且考虑到分区操作系统不提供像 malloc 这样的函数，因此只需要在配置文件中规定栈的大小，就可以保证分区内存使用量是定值。

3.3　基于 seL4 的分区调度

本节介绍在用户态创建一个分区调度任务来实现分区调度。已经有文献对用户态的调度进行过研究，指出用户态调度主要问题是开销较大，但是能够带来灵活性[9][10]、容易调试[11][12][13]等优势。分区调度的对象是 seL4 的线程，具体来说就是内核 TCB 对应的一个线程。用户态任务要想做到这一点，必须能够触发内核 TCB 的切换，即任务上下文的切换。seL4 的 suspend 和 resume 两个接口提供了这个功能。只要调度器选定了目标任务，并且当前正在运行的任务不是目标任务，只需要调用 suspend 和 resume 各一次就可以完成切换。一般来说，操作系统在内核里实现调度器[14]，一大原因就是可以利用内核处理时钟中断的便利。要在用户态实现调度器，首先要解决的就是调度时机问题[15]。seL4 提供了一种由用户进程注册中断并进行中断处理的方法。根任务创建一个额外的线程处理定时器时钟中断，并执行调度即可。以 x86 平台为例[16]，seL4 会同时使用两个时钟设备来满足这一要求。一个设备供内核调度器使用，另一个设备负责为用户态程序触发时钟中断，这两者并不矛盾，如图 6 所示。这样，用户态调度器就具备了第一个调度时机，即时钟中断处理后的调度时机。

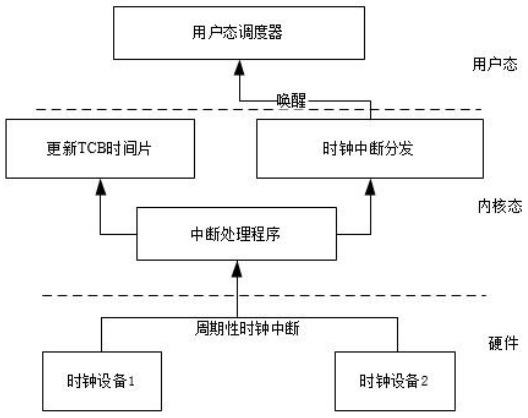

图 6　用户态程序处理时钟中断

第二个调度时机是 APEX 接口的调用返回之前。因为有的 APEX 会改变任务状态（例如 SUSPEND 接口），因此需要在返回之前立即进行一次调度。用户态调度器通过保证 suspend 和 resume 成对调用，从而让内核中只有一个可被调度的任务。这样就架空了内核调度器，从而避免了对用户态调度的干扰。

3.4　基于 seL4 的分区间通信方法研究

seL4 虽然不提供基于内核缓冲区的异步消息机制，但可以使用一个用户态的任务来充当内核缓冲区的角色，并进行消息的转发，从而实现异步的通信。此外，该线程在实施消息转发的同时，对分区间的通信进行权限检查[17]，从而实施分区间通信的控制。这个用户态任务可以用根任务来实现。基于根任务的分区间通信方法如图 7 所示。

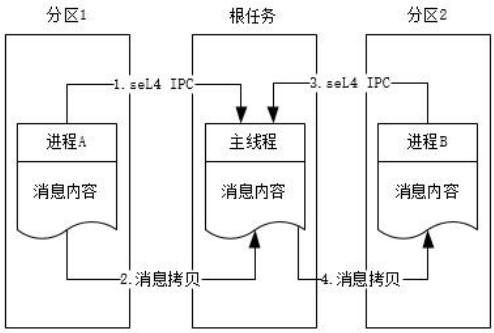

图 7　用户态分区间通信方法

由图 7 可知，如果一个分区的进程 A 需要给另一个分区的进程 B 发送消息，需要通过以下方式完成：首先，任务 A 通过端点向根任务发送消息，根任务接收到消息后，拷贝消息内容到消息缓冲区中，任务 A 的消息被响应，因此任务 A 继续运行；任务 B 所在分区被唤醒后，也向根任务发消息，请求接受刚才 A 发来的消息；根任务收到消息，通过 seL4 的 IPC 把消息发送给任务 B。以上就完成了一次完整的消息通信的过程。

4　seL4 多核和分区机制的结合

前面讨论的 seL4 多核设计与分区机制的研究是单独进行的，也即在单核 seL4 基础上分别加入多核的支持和分区机制。而本小节提出在多核 seL4 平台基础上引入分区机制，即在 seL4 微内核（BSP）启动之后，初始化其他的 AP、运行根任务、根任务运行在第一个启动的核心上，根任务负责完成分区的配置工作，最终的技术方案架构图如图 8 所示：

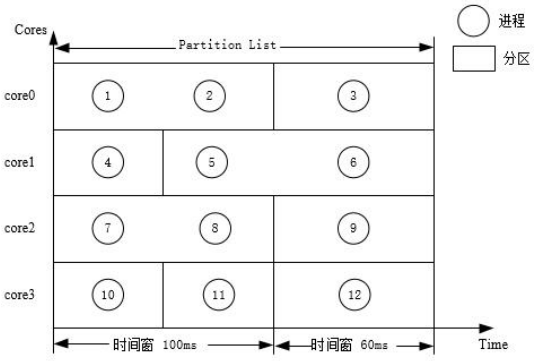

图 8　方案架构图

图 8 中横轴表示时间，纵轴表示核心数。图中并未标注根任务。在 qemu 启动过程中，通过-smp 参数，传入核心数，比如-smp 4，表示目前开了 4 个核心。每一个核心上面绑定一个分区列表（Partition List）。它们之间是一一对应的，即一个核心对应一个分区列表，对应关系在根任务中进行配置，每个核心的分区数目可以根据用户的需要自定义，但在系统运行过程中，分区配置信息一旦初始化好之后便不可更改。由于分区采用 seL4 上的 VSpace 来实现，这保证了分区之间的内存是不能互相访问的。同时也将保证某个核心上面的分区发生错误，比如 core1 上面的分区在运行过程中，发生错误不能传播，受影响的只是 core1，其他分区将会正常运转。这正体现了分区机制的容错和安全特性。

5　结束语

本文就 seL4 的多核调度与分区隔离进行了研究，设计并实现了在单核 seL4 的基础上分别加入多核和分区隔离的支持。最后将这两部分工作结合起来，提出了多核平台的分区机制方案，未来可在 qemu 模拟器上运行。

参考文献

[1] Klein G.,Andronick J.,Elphinstone K.,et al. seL4: formal verification of an operating system kernel[J]. Communications of the Acm, 2010, 53(6):107−115.

[2] Peters S., Danis A., Elphinstone K., et al. For a Microkernel, a Big Lock Is Fine[C]. Proceedings of the 6th Asia-Pacific Workshop on Systems.2015. New York, NY,USA: ACM, APSys'15, http://doi.acm.org/10.1145/2797022.2797042.

[3] Alves-Foss J., Oman P. W., Taylor C., et al. The MILS architecture for high-assurance embedded systems[J].Internationl journal of embedded systems, 2006, 2(3-4): 239-247.

[4] Prisaznuk P. J. ARINC 653 role in integrated modular avionics (IMA)[C]. 2008 IEEE/AIAA 27th Digital Avionics Systems Conference, 2008:1-E.

[5] Committee A. E. E., et al. Avionics application software standard interface part 1-required services[J].ARINC Document ARINC Specification 653P1-2, Aeronautical Radio, Inc., Annapolis, Maryland, 2006.

[6] Han S., Jin H.-W.Kernel-level ARINC653 partitioning for Linux[C]. Proceedings of the 27th Annual ACM Symposium on Applied Computing, 2012:1632-1637.

[7] DelangeJ.,Lec L.POK, an ARINC653-compliant operating system released under the BSD license[C]. 13th Real-Time Linux Workshop, 2011,10.

[8] AsbergM.,NolteT.Towardsa user-mode approach to partitioned scheduling in the seL4 microkernel[J]. ACM SIGBED Review, 2013, 10(3): 15-22

[9] Ford B.,SusarlaS.CPU inheritance scheduling[C]. OSDI 1996, 96:91 – 105.

[10] Lackorzyński A., Warg A., Völp M., et al. Flattening hierarchical scheduling[C].Proceedings of the tenth ACM international conference on Embedded software, 2012:93 – 102.

[11] Heiser A. L. G. FlaRe: Efficient Capability Semantics for Timely Processor Access[J], White Paper by NICTA and University of New South Wales (UNSW), 2015.

[12] Blagodurov Sergey,Fedorova Alexandra User-level scheduling on NUMA multicore systems under Linux[C]. Linux symposium 2011:81-91.

[13] Mercer C.W., Savage S., Tokuda H. Processor capacity reserves: An abstraction for managing processor usage[C]. Fourth Workshop on Workstation Operating Systems, 1993:129-134.

[14] Uhlig R., Neiger G., Rodgers D., et al. Intel virtualization technology[J]. Computer, 2005, 38(5): 48-56.

[15] Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. Linux Journal, 2014, 2014(239):2.

[16] Kaiser R.Combining partitioning and virtualization for safety-critical systems[J]. White Paper WP CPV, 2007,10:A4

[17] Klein G., Andronick J., Elphinstone K., et al. Comprehensive formal verification of an OS microkernel[J]. ACM Transactions on Computer Systems (TOCS), 2014,32(1):2

作者的联系电话和邮箱：

丁贵强　电话：18810303964

Email: 920398694@qq.com

王雷*　电话：13910666679

Email: wanglei@buaa.edu.cn

王鹿鸣　电话：13439166737

Email: wlm199558@126.com

康乔　电话：18610933136

Email: buaakq66@126.com