多核平台上针对 seL4 的分区机制研究

丁贵强[1] 王 雷[1] 王鹿鸣[1] 康 乔[2]
(北京航空航天大学计算机学院 北京 100191)[1] (北京航空航天大学软件学院 北京 100191)[2]

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,文中在目前只支持单核的 seL4 的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相结合的方案,实现了带分区机制的多核 seL4。最终将其运行在 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 TP316 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.09.010

Study of Partition Mechanism for seL4 on Multi-core Platform

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

Abstract In avionics and other embedded fields, the multi-core era has come. How to make full use of multiple cores has become the research hotspot in field of the system. At the same time, as the system integration is getting higher and higher, a hardware platform may need to run tasks with different security levels, which requires the operating system to provide isolation and protection for the application. In order to solve the two problems mentioned above, this paper added multi-core and partition isolation support on the basis of seL4 with a single core, and then put forward multi-core seL4 and partition mechanism to achieve a partition mechanism with multi-core seL4 running on the qemu simulator. The implementation of the partition mechanism is in accordance with the semantics of the ARINC653 standard.

Keywords Sel4, Multi-core, Partition, Embedded

1 引言

seL4(secure embedded L4)是由一款澳大利亚信息通信技术中心(NICTA)开发的微内核操作系统。本文在单核 seL4 的基础上加入多核支持,引入符合 ARINC653 标准的分区隔离,设计了多核分区系统。本文第 2 节介绍了 seL4 的多核系统设计;第 3 节介绍了分区主要标准,即 ARINC653 标准,然后在 seL4 中加入了分区隔离的支持;第 4 节提出了多核设计与分区机制相结合的方案;最后总结全文。

2 seL4 的多核设计

seL4 是第一个通过形式化方法被证明正确性的微内核操作系统[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,因此实现多核支持对于扩展 seL4 的应用范围而言十分必要。

多核的引入需要考虑数据竞争和多核调度器两方面的问题。数据竞争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。在解决数据竞争问题方面,普遍采用的方案是使用锁,例如 Linux,BSD 等均采用了细粒度的锁来保护内核的各类数据结构。而多核调度问题将在 2.3 节进行讨论。

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,其正确性易得到验证,是比较适合本文的方案。

大内核锁方案的总体设计如图 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,并设置该线程为当前核心上正在运行的线程。

到稿日期:2017-10-15 返修日期:2018-01-10 本文受国家自然科学基金(61672073,61272167)资助。
丁贵强 男,硕士生,主要研究方向为微内核操作系统;王 雷 男,博士,副教授,主要研究方向为操作系统,E-mail:wanglei@buaa.edu.cn(通信作者);王鹿鸣 男,主要研究方向为操作系统;康 乔 男,硕士生,主要研究方向为虚拟化。

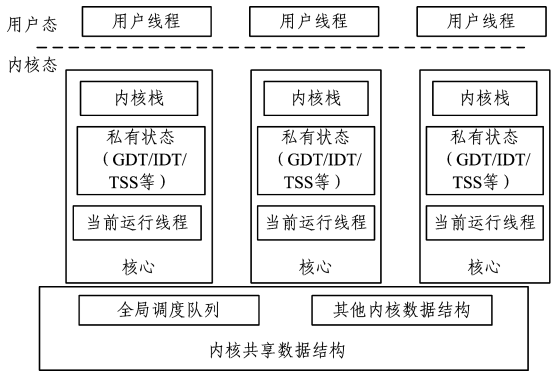

图 1　总体设计图

Fig. 1　System design

该方案的思路是将内核整体作为一个临界区,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。其主要优点在于对代码的改动较小,只需要在进入和退出内核时添加锁的获取和释放语句即可。由于 seL4 系统调用的执行流程较短,大内核锁对性能的影响是有限的。在贴近真实应用场景的性能测试中,大内核锁方案表现出的可扩展性与更细粒度的锁、RTM(Restricted Transactional Memory)锁等相近。因此,对于 seL4 微内核而言,本方案的性能是可以接受的[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 的多核启动流程来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 BSP 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数据结构,最后唤醒所有 AP,在各个 AP 上执行对 AP 的初始化。整个流程如图 2 所示。

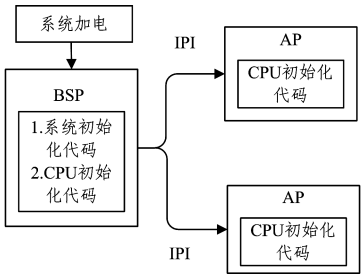

图 2　x86 多核启动流程

Fig. 2　x86 multi-core startup process

在 seL4 中实现多核启动需要解决 3 个问题:1)如何唤醒其他核心;2)如何设置各个核心运行时所需的内核栈;3)针对当前 AP,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。通过向其他的 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的执行代码。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 发送初始化 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与当前核心相关,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,并将其设置为当前正在执行的线程。之后当退出内核空间时,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,即如何使其他的核心执行代码,这一点只能通过中断来实现。通过产生中断,可使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。有两种中断适用于这一应用场景: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其他的核心上执行代码,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,影响了调度效率。因此,采用时钟中断来实现调度成为首选。每当核心产生时钟中断时,系统会进入到多核调度器代码中,调度器会为当前核心挑选一个线程投入运行。由于每个核心都会产生时钟中断,因此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提高,人们期望在同一硬件平台上集成多个应用程序,从而提高集成度并节约成本,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:1)MILS(Multiple Independent Levels of Security/Safety)[3];2)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。本文研究了 seL4 支持 ARINC653 标准的可行性,设计了一套基于 seL4 实现分区隔离的方案,以帮助 seL4 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:分区(Partition)和进程(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。下面对这几个方面进行简要介绍。

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容错,即一个分区的失效不能传播到其他分区中。

ARINC653 分区的主要属性如表 1 所列。

表 1 分区的主要属性

Table 1 Main properties of partition

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

空间隔离是指分区不能访问其他分区的内存，且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，如一般不提供类似 malloc 的系统调用；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，为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同，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。对于时间隔离，ARINC653 制定了一种独特的二级调度标准，如图 3 所示。只有当时间落在分区对应的窗口内时，对应的分区才处于活跃态，分区内的进程才能运行。宏观来看，调度具有周期性，一个周期被称作主时间窗。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)，也即图 3 所示的时间配置在系统启动后不会再发生改变。

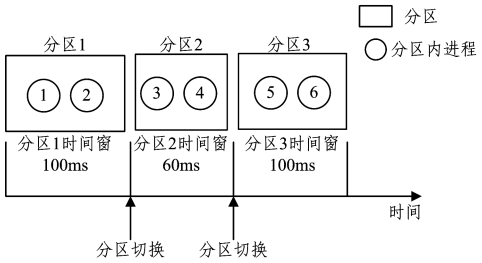

图 3 分区调度

Fig. 3 Partition scheduling

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，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 所列。

表 2 进程的主要属性

Table 2 Main attributes of process

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，如 pok[7]，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需要指定相应的端口。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接收端口，但不是所有的发送端口、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的消息请求时，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接收端口中，否则丢弃该消息。操作系统并不关心消息的具体内容，只是简单地进行消息的拷贝。

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，可以使用 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。

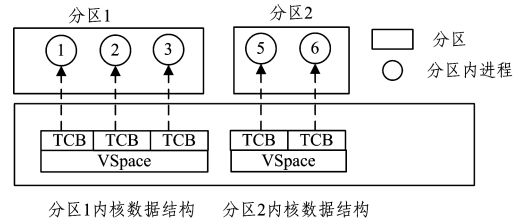

图 4 分区与进程在 seL4 中的表示

Fig. 4 Representation of partitions and processes in seL4

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，该层即为一个进程(即 seL4 的根任务)。除根任务的主线程外，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相同，而调度线程与分区内进程的图示相同。

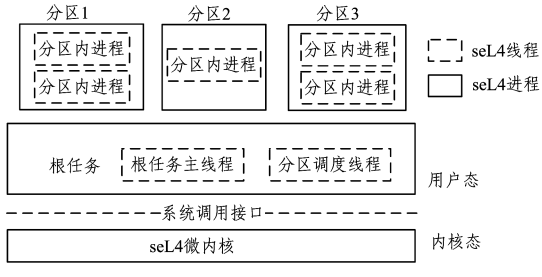

图 5 用户态分区管理层

Fig. 5 User-space partition management

seL4 内核启动之后，首先运行根任务。根任务与其他分区实际上是平等的：它们在 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

务与根任务通信，根任务需要创建一个同步端点（Endpoint），以提供 APEX（APplication EXecutive）接口的服务。当用户进程调用 APEX 接口时，实际上是通过 seL4 的 IPC 给根任务主线程发送消息来完成。由于发送消息需要使用同步端点，因此根任务必须在初始化时创建该端点。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。调度线程创建完毕后，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。针对第一点，由于使用了 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不能互相访问。当发生分区切换时，seL4 内核会执行页表的切换。针对第二点，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即确定的，并且考虑到分区操作系统不提供类似 malloc 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

本节介绍了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研究，指出用户态调度的主要问题是开销较大，但是具有灵活性[9-10]、容易调试[11-13]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，就必须能够触发内核 TCB 的切换，即任务上下文的切换。seL4 的 suspend 和 resume 两个接口提供了该功能。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，则只需要调用 suspend 和 resume 各一次就可以完成切换。一般来说，操作系统在内核中实现调度器[14]，其主要原因就是可以利用内核处理时钟中断。要想在用户态实现调度器，首先要解决调度时机问题[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。

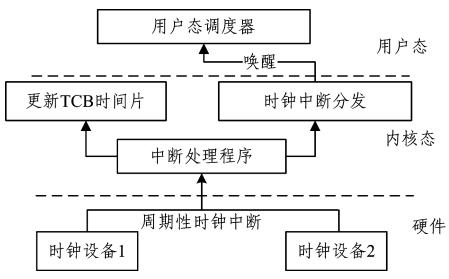

图 6　用户态程序处理时钟中断

Fig. 6　User-mode program processing clock interruption

第二个调度时机在 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），所以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，使得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，该用户态任务可以通过根任务实现。基于根任务的分区间通信方法如图 7 所示。

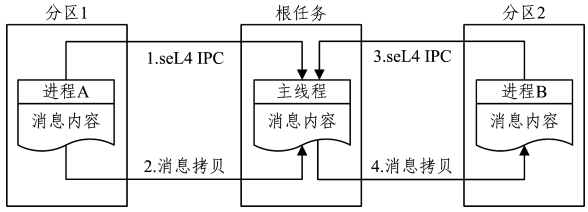

图 7　用户态分区间通信方法

Fig. 7　User mode inter-partition communication method

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，根任务接收到该消息后，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，也向根任务发送消息，请求接收 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。以上即完成一次完整的消息通信的过程。

4　seL4 多核和分区机制的结合

前文讨论的 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 的基础上分别加入多核的支持和分区机制。而本节提出在多核 seL4 平台的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 AP，运行根任务。根任务运行在第一个启动的核心上，负责完成分区的配置工作，最终的技术方案架构如图 8 所示。

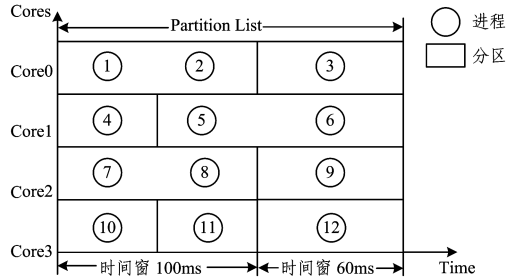

图 8　方案架构图

Fig. 8　Scheme architecture

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，如-smp 4，表示目前开了 4 个核心。每一个核心上绑定一个分区列表（Partition List），它们之间是一一对应的关系，即一个核心对应一个分区列表，对应关系在根任务中配置。每个核心的分区数目可以根据用户的需求自定义，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，保证了分区之间的内存

不能互相访问;同时也将保证某个核心上的分区发生错误时不会传播,如 Core1 上的分区在运行过程中发生错误时不能传播,受影响的只是 Core1,其他分区仍会正常运转,这正体现了分区机制的容错和安全特性。

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 模拟器上运行该方案。

参 考 文 献

[1] KLEIN G,ANDRONICK J,ELPHINSTONE K,et al. seL4:formalverification 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. Newyork. NY,USA:ACM,APSys'15,2015.

[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.

[5] COMMITTEE A E E,et al. Avionics application software standard interface part 1required services[EB/OL]. https://standards. globalspec. com/std/280829/arinc-653p1.

[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] DELANGE J,POK L L. An ARINC653-compliant operating system released under the BSD license[C]// 13th Real-Time Linux Workshop. 2011.

[8] ASBERG M,NOLTE T. Towardsauser-mode approach to partitioned scheduling in the seL4 microkernel[J]. ACM SIGBED Review,2013,10(3):15-22.

[9] FORD B,SUSARLA S. CPU inheritance scheduling[C]// Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation(OSDI'96). 1996:91-105.

[10] LACKORZYN′SKI 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,LYONS A,VANGE M,et al. FlaRe:Efficient Capability Semantics for Timely Processor Access[EB/OL]. https://people. mpi-sws.org/~bbb/papers/pdf/preprint-FlaRe. pdf.

[12] SERGEY B,ALEXANDRA F. User-level schedulingon NUMA multicore systems under Linux[EB/OL]. http://citeseerx. ist. psu. edu /viewdoc/download?doi=10. 1. 1. 369. 9422&rep=rep1&type=pdf.

[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] NICTA. The seL4 microkernel[EB/OL]. https://github. com/seL4/seL4.

[17] KLEIN G,ANDRONICK J,ELPHINSTONE K,et al. Comprehensive formal verification of an OS micro-kernel[J]. ACM Transactions on Computer Systems(TOCS),2014,32(1):32-102.

(上接第 69 页)

[26] FAN J C,ZHANG W Y,LIANG Y Q. Decision tree classification algorithm based on Bayesian method[J]. Journal of Computer Applications,2005,25(12):2882-2884. (in Chinese)
樊建聪,张问银,梁永全. 基于贝叶斯方法的决策树分类算法[J]. 计算机应用,2005,25(12):2882-2884.

[27] FRANK A,ASUNCION A. UCI Machine Learning Repository[DB/OL]. http://archive. ics. uci. edu/ml/Irvine,CA:University of California,School of Information and Computer Science.

[28] YANG L Y,ZHANG J Y,WANG W J. Selecting and Combining Classifiers Simultaneously with Particle Swarm Optimization[J]. Information Technology Journal,2009,8(2):241-245.

[29] SINGH R G,PANDEY A. The Impact of Randomization on Circular-Complex Extreme Learning Machine for Real Valued Classification Problems[J]. International Journal of Computer Applications,2014,103(2):1-7.

[30] LIPITAKIS A D,ANTZOULATOS G S,KOTSIANTIS S,et al. Integrating global and local boosting[C]// 2015 6th International Conference on Information,Intelligence,Systems and Applications(IISA). IEEE,2015:1-6.

[31] RAHMAN A,VERMA B. A novel ensemble classifier approach using weak classifier learning on overlapping clusters[C]// International Joint Conference on Neural Networks. IEEE,2010:1-7.

[32] COELHO A L V,NASCIMENTO D S C. On the evolutionary design of heterogeneous bagging models[J]. Neuro Computing,2010,73(16):3319-3322.

[33] CHEN J,JI S,CERAN B,et al. Learning subspace kernels for classification[C]// Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM,2008:106-114.

[34] DO T N,POULET F. Enhancing svm with visualization[C]// International Conference on Discovery Science. Springer Berlin Heidelberg,2004:183-194.

[35] QUINLAN J R. Bagging,boosting,and C4. 5[C]// Association for the Advancement of Artificial Intelligence. 1996:725-730.

[36] CLARK P,BOSWELL R. Rule induction with CN2:Some recent improvements[C]// European Working Session on Learning. Springer Berlin Heidelberg,1991:151-163.

[37] JO H,NA Y,OH B,et al. Attribute value taxonomy generation through matrix based adaptive genetic algorithm[C]// 20th IEEE International Conference on Tools with Artificial Intelligence. IEEE,2008,1:393-400.

[38] SAEED A A,CAWLEY G C,BAGNALL A. Benchmarking the semi-supervised naïve Bayes classifier[C]// International Joint Conference on Neural Networks. IEEE,2015:558-561.