库操作系统的研究及其进展

舒红梅[1] 谭 良[1,2]

(四川师范大学计算机科学学院 成都 610101)[1] (中国科学院计算技术研究所 北京 100190)[2]

摘 要 最早的库操作系统(Library OS,LibOS)基于外内核架构,目的是验证在用户空间管理系统资源的可行性和高性能性。但是,由于外内核还停留在研究上,实际应用中仍以宏内核和混合内核为主,因此 LibOS 一开始并没有引起学术界和产业界的过多关注。伴随云计算的快速发展和物联网的兴起,为了构建安全高效的 Unikernel 云服务和物联网微服务,LibOS 成为了新的研究热点。首先总结了 LibOS 的基本定义和基本特点;然后提出了 LibOS 分类模型;接着总结了 LibOS 的系统架构,并详细阐述了 LibOS 的关键技术,包括 LibOS 内核基中的线程管理、CPU 调度和虚拟内存管理以及 LibOS 功能系中的网络服务功能、文件 I/O 功能和设备访问功能等;最后结合已有的研究成果,探讨了 LibOS 面临的问题和挑战。

关键词 库操作系统,Unikernel,云计算,微服务

中图法分类号 TP316 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.11.004

Research and Development of Library Operating System

SHU Hong-mei[1] TAN Liang[1,2]

(College of Computer Science,Sichuan Normal University,Chengdu 610101,China)[1]

(Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100190,China)[2]

Abstract The earliest library operating system (LibOS) is based on exokernel,and its purpose is to verify feasibility and high performance for management system resources in user space. However,exokernel is still in the study. Macro kernel and Hybrid kernel are the main operating system architectures in actual application. So LibOS didn't attract much attention from academia and industry at the beginning. As the rapid development of cloud computing and the rise of the Internet of Things,in order to build secure and high-performance Unikernel which is a kind of special micro-service,LibOS has become a new research hotspot. Firstly,the basic definition and features of LibOS were introduced,and the classification model of LibOS was put forward. Secondly,the architecture of LibOS was proposed and the key technologies of LibOS were described in detail,including thread management,virtual CPU scheduling,virtual memory management in LibOS kernel base,and network service,the disk file I/O and devices access in LibOS functions,etc. Finally,based on the existing research results,this paper discussed the problems and challenges of LibOS.

Keywords Library operating system(LibOS),Unikernel,Cloud computing,Micro services

1 引言

操作系统[1]自 20 世纪 50 年代产生并发展到现在,其内核[2](kernel)的发展经历了宏内核[3](Monolithic Kernel)、微内核[4-6](Micro Kernel)、混合内核[2](Hybrid Kernels)到外内核[7-12](Exokernel)的研究变化。这些结构都是操作系统根据实际应用场景的不同需求进化而来,并不断朝着专业、精简、安全、高效、灵活、可扩展的方向发展[13-18]。现阶段,在实际应用的操作系统中,采用宏内核架构的有 Linux,Unix,FreeBSD 等,采用微内核架构的有 QNX,MINIX 和 Mach 等,也有许多现代操作系统使用混合内核,例如 Windows 7 和 BeOS 等。而 Exokernel 是操作系统设计的一种实验性方法,不同于以上内核,它将系统资源的管理和保护分离,内核只提供最原始的硬件保护和复用,而将硬件资源的管理权限交给应用程序的开发者,使其能够确定如何最有效地利用每个特定程序的可用硬件[8]。外内核本身非常小,但是它需要库操作系统(LibOS)为应用程序开发人员提供操作系统的完整常规功能。应用程序通过依赖 LibOS 提供的底层硬件资源抽象接口直接对硬件进行操作,同时开发者必须拥有较强的硬件管控能力,以防止硬件被滥用,通过这种方式就有可能解决宏内核性能损失的难题[7,19]。但是,由于外内核还停留在研究上,实际应用中仍然以宏内核和混合内核为主,因此库操作系统一开始并没有引起学术界和产业界的过多关注。

随着云计算[20-21]的快速发展和虚拟化技术[22-23]的成熟,

到稿日期:2017-09-10 返修日期:2017-10-31 本文受国家自然科学基金(61373162),四川省科技支撑项目(2014GZ0007)资助。

舒红梅(1992-),女,硕士生,主要研究方向为信息安全,E-mail:shm.1992@foxmail.com;**谭 良**(1972-),男,博士,教授,CCF 高级会员,主要研究方向为可信计算、网络安全、云计算和大数据处理,E-mail:tanliang2008cn@126.com(通信作者)。

我们已经看到大量的项目和技术利用了云提供的灵活性。目前,云正在朝着单一核心化方向发展,并希望通过使用最少的资源来实现安全、高效的应用服务。现在,云环境中的资源都是以虚拟机的形式为用户提供服务,许多服务都是依托于虚拟机的形式存在。然而,虚拟机上安装的操作系统一般都是针对物理机而设计的通用宏内核或混合内核操作系统,比如 Linux,Windows 等,其中存在大量不必要的系统组件,不仅占用了宝贵的资源空间,影响了软件服务的运行效率,还增加了系统面临的安全威胁(比如云环境中并不需要的 USB 系统组件就增加了可能的攻击漏洞)。因此,人们对部署在云平台上为租户提供各种应用服务的虚拟机提出了越来越多的要求,如安全性的保证,虚拟机专业化、可定制的需求,尽可能少的资源占用(最大限度地提高资源利用率),以及弹性可伸缩的需求服务、高性能的运行效率、额外零开销(用户不必为没使用的消耗付费)等[24-26]。基于库操作系统的 Unikernel[25-32]是能够满足以上应用场景需求的热门技术,现已成为专家学者们越来越关注的研究课题。通过库操作系统构建出的密封、用途固定的机器镜像(即 Unikernel)可以直接运行在虚拟机管理程序(hypervisor)或硬件上,而不依赖于 Linux 或 Windows 这样传统的操作系统。

然而,当前库操作系统种类繁多、可用性差,正在成为构建 Unikernel 及微服务的瓶颈。库操作系统的最初原型是 Exokernel[8]和 Nemesis[9],在库操作系统不断演进的过程中,分化出了许多不同的发展方向,目前专注于构建 Unikernel 的方法已不少于 12 种,主要包括 Drawbridge[33-34],IncludeOS[25,35],MirageOS[26-28,36],HaLVM[37],Graphene[38-39],ClickOS[40],Clive[41],LING[42],HermitCore[43-44],Runtime.js[45],OSv[46-47]和 Rump Kernels[48-50]。对于这些纷繁复杂的库操作系统,需要厘清一些基本的问题:如何将它们进行有效的分类,它们各自具有什么特点,采用怎样的系统架构,需要用到哪些关键技术,还存在哪些不足,等。面对以上问题,目前国内外尚未有详细且全面介绍库操作系统的综述论文。因此,综述库操作系统的研究进展有着重要的意义。

本文第 2 节介绍库操作系统的基本概念,包括其基本定义和基本特点;第 3 节根据 LibOS 运行的环境不同,提出了库操作系统的分类模型;第 4 节详细阐述库操作系统的关键技术,包括 LibOS 内核基中的线程管理、CPU 调度和虚拟内存管理,以及 LibOS 功能系中的网络服务功能、文件 I/O 功能和设备访问功能等;最后结合已有的研究成果,第 5 节探讨了库操作系统面临的问题和挑战。

2 库操作系统的基本概念

第一个库操作系统是由麻省理工学院(MIT)并行与分布式操作系统工作小组(Parallel and Distributed Operating Systems Group)的 Engler 和 Kasshoek 等在外内核操作系统 Aegis 的基础上实现的,称为 ExOS[8]。目前,学术界和产业界尚没有对库操作系统形成统一的定义,其基本特点也缺少总结,因此本节中将介绍其基本概念,包括基本定义和基本特点。

2.1 库操作系统的基本定义

定义 1(库操作系统,Library Operating System,library OS 或 LibOS) 库操作系统是根据某类应用的特殊需求,由某一高级编程语言将原本属于操作系统内核的某些资源管理功能,如文件磁盘 I/O、网络通信等,按照模块化的要求,以库形式提供给应用程序的特殊操作系统。它能代替操作系统内核合理地管理和控制所涉及的计算机资源,并将所涉及的计算机资源直接暴露给应用程序,让应用程序直接访问底层(虚拟)硬件,以便应用程序能够高效地运行。特别地,它与应用程序在编译时被链接到一起,形成一个只有单地址空间的二进制文件并工作在应用层,是构建 Unikernel 的一个必不可少的系统组件。

由定义 1 可以看出,库操作系统的概念包含 3 方面含义,具体如下:

(1)库操作系统本质是由某一高级编程语言实现的一系列库(libraries),这些 libraries 都是被模块化了的软件栈(stack),用以实现原操作系统内核中某些计算机资源的管理功能。

(2)库操作系统是一个特殊的操作系统,仍然具有传统操作系统的基本功能,如线程管理、CPU 调度以及虚拟内存管理等。但与传统操作系统不同的是,为了减少系统抽象,库操作系统工作在应用层,通过对所涉计算机资源的管理与调度功能,让应用程序直接访问所涉计算机资源,为应用程序提供所需的运行环境(run time)。

(3)库操作系统的典型应用是结合虚拟化技术,构建出适用于虚拟化平台以及嵌入式平台的 Unikernel 系统。

2.2 库操作系统的基本特点

根据定义 1,并通过对已有库操作系统的具体分析,总结出 LibOS 具有如下基本特点。

(1)模块化(modularization)

所谓模块化,是指将操作系统的功能结构化,按模块划分,以类库(library)来实现和封装系统的基本功能。模块化可以抽象出软件内部的实现细节,限制单个源代码更改的范围,并方便对各个部分做出修改和重用,满足其专业定制的需求,灵活组装出精简、优化的库操作系统。

(2)专用化(customization)

所谓专用化,是指开发人员可以根据自己对应用服务的特殊需求改写库,实现自定义抽象,省略不必要的抽象,跨设备驱动程序和应用逻辑执行全系统优化,剔除不需要的系统组件,从而构建出高度专业化的操作系统,满足其安全性和高效性。

(3)单地址空间(single address space)

所谓单地址空间,是指应用程序与特定的驱动、核心 library 运行在一起,共享同一内存页表,不存在运行时状态上下文切换的额外开销,因此可以提高程序的运行速率,实现更高效的应用服务。

3 库操作系统的分类模型

按照库操作系统为应用程序提供的资源管理功能来分类,是最自然和相对简单的分类方法。比如可以把库操作系统分为:网络服务 LibOS、文件系统 LibOS、用户态进程管理 LibOS、进程间通信 LibOS、同步互斥内核琐 LibOS 以及内存管理 LibOS 等[7]。为了进一步厘清库操作系统与所在运行

环境的依赖关系，我们根据 LibOS 的不同运行环境将其分为两类：1）基于主机环境的 LibOS；2）基于虚拟环境的 LibOS。

3.1　基于主机环境的 LibOS

所谓基于主机环境的 LibOS，是指依赖主机操作系统且运行在主机操作系统应用空间的库操作系统，如图 1 所示。

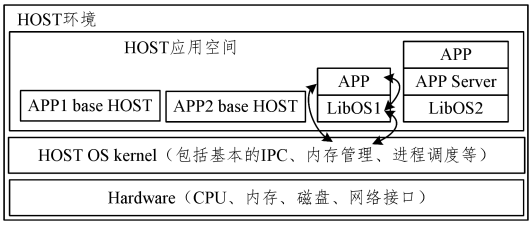

图 1　基于主机环境的 LibOS

Fig. 1　LibOS based on host environment

在图 1 中，底层是硬件资源。中间层是宏内核、微内核或外内核操作系统，包括基本的 IPC（进程间通信）、内存管理、进程管理和异常处理等基本功能，负责对硬件资源实现最基本的分配、撤销和管理，并通过安全绑定策略来允许库操作系统对资源的绑定使用，从而为互不信任的应用程序提供隔离保护。上层是不同功能的库操作系统和应用程序，库操作系统和应用程序作为一个整体运行在同一地址空间来实现应用层级别的功能。对于基于主机环境的 LibOS，有两点需要强调：1）HOST OS kernel 既可以是微内核或外内核，也可以是宏内核，但在微内核或外内核上实现 LibOS 相对容易，因为在微内核或外内核操作系统上已经实现了资源管理和保护的分离，尤其是外内核；2）HOST OS kernel 上包含两种应用，即 APP base HOST 和 APP base LibOS，而 APP base LibOS 又分两种情况，一种是 APP 直接调用 LibOS，另一种是 APP 通过 APP Server 调用 LibOS。

此类 LibOS 在早期的研究中成果较多。Kasshoek 等在外内核操作系统 Aegis 的基础上实现了库操作系统 ExOS[8,10]；微软基于混合内核 Windows 7 重构了 Drawbridge[33] LibOS；随后，Baumann 等在 Drawbridge 的基础上进行扩展，实现了一个既可运行 Windows 应用程序又可运行 Linux 应用程序的库操作系统 Bascule[18]；2014 年出现了基于 Linux 内核并兼容 Linux 多进程应用程序的 Graphene[38]。另外，国内兰州大学的学者设计并实现了基于外内核操作系统 JOS 的文件系统 LibOS[7] 和时间子系统 LibOS[12]。

3.2　基于虚拟环境的 LibOS

所谓基于虚拟环境的 LibOS，是指依赖于底层的 Hypervisor[22-23] 并运行在虚拟环境应用空间的库操作系统，如图 2 所示。

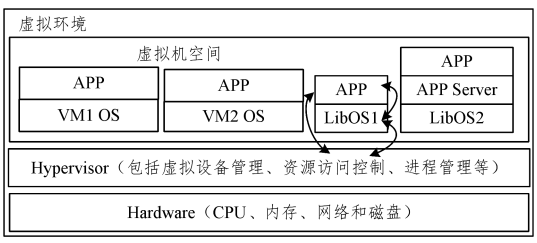

图 2　基于虚拟环境的 LibOS

Fig. 2　LibOS based on virtual environment

在图 2 中，底层是硬件资源。中间层是虚拟机管理程序（Hypervisor），或称作虚拟机监视器（Virtual Machine Monitor，VMM），实现虚拟设备管理、资源访问控制和进程管理等基本功能。Hypervisor 加载虚拟机客户端的操作系统时，会给每一台虚拟机分配适量的内存、CPU、网络和磁盘等硬件资源，并协调服务器上包括磁盘和内存在内的所有硬件资源的访问，同时在各个虚拟机之间施加防护。上层是不同类型的虚拟机，包括两种：1）一般虚拟机，APP 运行在通用操作系统之上；2）轻量级虚拟机，APP 运行在 LibOS 上，称为 Unikernel。

目前，由 Xen 社区主导开发的 MirageOS[26-28,36] 是基于虚拟环境的 LibOS 最具代表性的研究项目，另外 IncludeOS[25,35]，HaLVM[37]，ClickOS[40]，Clive[41]，LING[42]，HermitCore[43-44]，Runtime.js[45]，OSv[46-47] 和 Rump Kernels[48-50] 等都是基于虚拟环境的 LibOS 研究项目。国内目前还没有针对虚拟化平台的 LibOS 研究项目。

4　库操作系统的系统架构和关键技术

通过对已有 LibOS 项目的分析，我们总结出 LibOS 的通用系统架构，如图 3 所示。

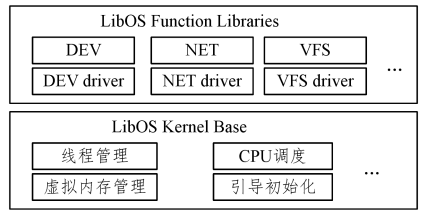

图 3　LibOS 系统架构

Fig. 3　Architecture of LibOS

从图 3 可以看出，LibOS 的通用系统架构被划分为两个逻辑层：内核基和功能系。内核基是每个库操作系统运行的基础，包含一些最基本的功能，如线程管理、CPU 调度、虚拟内存管理以及引导初始化等。功能系则是支持应用程序运行所依赖的最小核心组件，包含协议库和驱动程序。上层应用程序通过功能库的可调用函数实现对底层物理硬件的快速访问。下文将详细分析 LibOS 内核基以及功能系中的关键技术。

4.1　LibOS 的内核基

LibOS Kernel Base，即 LibOS 内核基，是 LibOS 的核心，一般均具有线程管理、CPU 调度、虚拟内存映射以及引导初始化等基本功能。当然，还有一些特殊的功能与 LibOS 的具体应用有关。例如，对于文件 LibOS，还应具有进程特权等级等功能。鉴于篇幅，下面仅对 LibOS 的线程管理、CPU 调度、虚拟内存映射等进行详细阐述。

4.1.1　线程管理

线程管理是 LibOS 的基本功能之一。LibOS 必须协同主机环境的 OS 或虚拟平台的 Hypervisor 一起完成对 APP base LibOS 的线程管理。对于主机环境的 LibOS，从 HOST OS kernel 的视角来看，无论是 APP base HOST 还是 APP base LibOS，作为一个基本的调度单位，其地位是平等的。但是在实际运行环境中，APP base HOST 和 APP base LibOS

是不同的,前者的运行托管环境是 HOST,而后者的运行托管环境是 HOST 和 LibOS,因此,当运行 APP base LibOS 时,不仅需要 HOST 环境的硬环境上下文,还需要 LibOS 的软环境上下文。对于虚拟环境的 LibOS,其具体情况与主机环境的 LibOS 基本一致。因此,无论在主机环境还是在虚拟环境,LibOS 均需要线程管理功能辅助底层操作系统对 APP base LibOS 的调度和正常运行。除此以外,为了保证 APP base HOST 和 APP base LibOS 对共享资源的访问,以及优先运行高优先级的线程,一些 LibOS 的线程管理功能还包含线程优先级以及对共享资源访问时的互斥和同步。

当然,不同类型的 LibOS,其线程管理功能的实现方式和复杂程度不仅依赖于 LibOS 线程管理功能本身的复杂程度,而且依赖于底层软环境。特别地,基于主机环境的 LibOS 依赖于主机 OS,基于虚拟环境的 LibOS 依赖于底层虚拟平台。例如,ExOS 是一个典型的基于主机环境的 LibOS,其底层操作系统是 exokernel 类的 JOS,由于 JOS 已经实现了底层资源管理与保护的分离,因此 ExOS 线程管理是通过系统调用直接沿用 JOS 线程管理,即 APP base ExOS 线程上下文就是由 JOS 产生的线程上下文,APP base ExOS 线程优先级以及对共享资源访问时的互斥和同步方法均由 JOS 提供,因此实现较为简单。而对于 rump kernel 这一典型的基于虚拟环境的 LibOS,其线程管理要复杂得多。APP base rump kernel 线程的硬上下文由 HOST 创建,而软上下文则在 rump kernel 本地客户端创建[49]。对于对共享资源的访问,rump kernel 线程定义了一组超级调用接口,如 pthread_mutex_lock() 和 msleep() 等,这些接口通过封装底层操作系统的同类接口,提供了互斥锁、读/写锁和条件变量原语。

值得注意的是,无论是基于主机环境的 LibOS 还是基于虚拟环境的 LibOS,如果需要支持多种底层环境,则需要分别实现针对不同环境的线程管理功能。例如,rump kernel 既要支持 Xen,又要支持 KVM,则其必须分别实现针对 Xen 和 KVM 的线程管理功能。

4.1.2 CPU 调度

CPU 调度是 LibOS 的另一个基本功能。LibOS 必须协同主机环境的 OS 或虚拟平台的 Hypervisor 将当前需要执行的 APP base LibOS 线程调度到 LibOS 的 CPU,并形成正确的 CPU 上下文。一方面,APP base LibOS 与运行环境中的其他应用程序一样,作为一个基本的调度单位,参与运行环境的 CPU 调度;另一方面,如果按照某种策略为 LibOS 分配了多个虚拟 CPU 时间片,则在这多个虚拟 CPU 时间片内,就由 LibOS 协同底层环境的操作系统或 Hypervisor 按预定的策略进行虚拟 CPU 调度,这是 LibOS 支持多应用的基础;再一方面,当有更高优先级的 APP base LibOS 线程需要执行或 APP base LibOS 线程中有异步事件发生时,LibOS 应抢占当前线程 CPU 并执行相应的处理程序,然后将控制权返回到原始线程;最后,无论在主机环境还是虚拟环境中,LibOS 的 CPU 调度都需要考虑底层物理 CPU 是单核还是多核,如果物理 CPU 是多核,LibOS 还需要协同主机环境的 OS 或虚拟平台的 Hypervisor 一起将 APP base LibOS 线程调度到多核物理 CPU 上。

当然,不同类型的 LibOS,其 CPU 调度功能的实现方式和复杂程度不仅依赖于 LibOS CPU 调度功能本身的复杂程度,还依赖于底层环境。无论是主机环境还是虚拟环境,在引导 LibOS 之前,通常都需要通过静态或动态的配置方式为 LibOS 分配一个或多个虚拟 CPU,如果只分配一个虚拟 CPU,则这一个虚拟 CPU 调度主要由主机环境的 OS 或虚拟平台的 Hypervisor 完成;如果分配多个虚拟 CPU,这多个虚拟 CPU 则必须由 LibOS 协同主机环境的 OS 或虚拟平台的 Hypervisor 一起完成。例如,ExOS 的 CPU 调度是按照 ExOS 的调度策略通过系统调用直接沿用 JOS 的 CPU 调度方法;而 rump kernel 支持虚拟 CPU 的调度,最开始 rump kernel 维护一个空闲的虚拟 CPU 列表,通过在列表中输入一个条目来完成分配,并通过将其放回到列表中来完成释放,而且虚拟 CPU 的数量可以根据每个应用程序的场景进行配置,这种方式适用于单处理器主机。然而,在具有多 APP base rump kernel 线程的多核 CPU 系统上,全局列表会导致高速缓存争用和锁争用,这会导致 APP base rump kernel 线程在多虚拟 CPU 系统多处理器运行的时间大约是单处理器的 3 倍[49]。为了优化多虚拟 CPU 的情况,开发了一种改进的 CPU 调度算法,即在 rump kernel 中,当主机线程 A 放弃 rump kernel 虚拟 CPU 时,主机线程 B 可以在不同的物理 CPU 上获取相同的 rump kernel 虚拟 CPU。改进的 CPU 调度算法在 sys/rump/librump/rump kern/scheduler.c 中以代码实现,且该改进算法在双 CPU 多线程情况下的性能是最初算法的 4 倍[49]。

4.1.3 虚拟内存管理

与线程管理、CPU 调度一样,虚拟内存管理也是 LibOS 的一个基本功能。LibOS 必须协同主机环境的 OS 或虚拟平台的 Hypervisor 一起完成对 APP base LibOS 的虚拟内存管理,实现 APP base LibOS 的虚拟内存地址空间到物理内存的映射。LibOS 的虚拟内存管理与主机操作系统或 Hypervisor 不同,因为 LibOS 不使用(也不需要)硬件 MMU。为了保证虚拟内存管理的性能,LibOS 的虚拟内存管理中通常没有页面保护或页面错误等功能,因为通过用户空间去处理页面保护和页面错误会大大降低内存访问的性能。

目前,无论是基于主机环境的 LibOS 还是基于虚拟环境的 LibOS,其虚拟内存映射功能绝大多数采用在 LibOS 中定义相应的虚拟内存数据结构与底层 OS 或 Hypervisor 中的硬件 pages 进行映射。例如,在 ExOS 中,定义了 GlobalDescriptorTable 和 PageTable 数据结构,通过系统调用实现虚拟内存到物理内存的映射。而 rump kernel 的虚拟内存管理的实现更加复杂和强大。rump kernel 通过 struct vmpage 数据结构描述的页面与硬件 pages 相对应,通过超级调用实现了页面映射器、内存分配器和页面守护进程。页面映射器的主要功能是将要读取的内存空间由主机内核映射到客户端的地址空间,如图 4 所示。user proc 作为客户程序,当需要访问文件系统时,则通过主机内核的 puffs[50] 接口访问提交 I/O 请求,该 I/O 请求通过 libpuffs 发送到 rump kernel,rump kernel 将

file 映射到 user proc 的内存空间。如果出现页面错误，则该错误在主机内核中被解决。内存分配器实现页面级别上的内存分配，rump kernel 将 Net BSD 内核中的内存分配器作为 rump kernel 内存分配器。而页面守护进程是 rump kernel 虚拟内存管理的一种补偿机制，当可用内存将要耗尽时，rump kernel 将调用页面守护进程 pagedaemon。pagedaemon 本质上是一个内核线程，用以定位和释放内存资源，而这些资源在不久的将来很可能被使用。

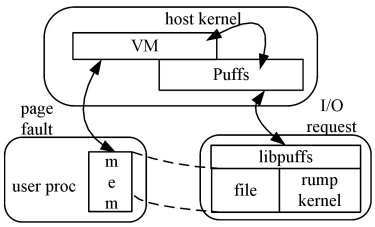

图 4　rump kernel 上的内存映射支持

Fig. 4　Memory mapping on rump kernel

4.2　LibOS 功能系

LibOS 功能系的主要目标是为 APP base LibOS 提供更高性能的计算机资源管理功能接口，例如网络服务功能、文件 I/O 功能以及设备访问功能等。LibOS 功能系通常包括相应的驱动、功能库和 API 接口。APP base LibOS 在应用空间调用 LibOS 功能系的 API 接口，就可以直接运行在硬件或者 Hypervisor 上，从而更加高效地利用计算机资源。目前，实现最多的功能系包括网络服务功能、文件 I/O 功能以及设备访问功能等。

4.2.1　网络服务功能

网络服务是主机环境或虚拟环境使用得最为频繁的功能，一般包括网络协议和驱动程序两部分。传统宏内核操作系统将网络服务的全部功能放在内核中，而微内核操作系统则是把网络协议功能部分作为驻留服务程序放在用户空间，只是把驱动功能部分放入微内核。这种处理方式尽管提升了网络服务的可靠性和安全性，但是降低了网络服务的性能。为了既保证可靠性和安全性，又能获得网络服务的高性能，可通过 LibOS 来实现网络服务。当在 LibOS 中实现网络服务功能时，网络协议功能部分和驱动功能部分均在应用空间实现，应用程序调用 LibOS 网络服务功能 API，不需要运行环境的上下文切换，从而提高了网络通信的速度。

当然，不同类型的 LibOS，其网络服务功能的实现方式和复杂程度不仅依赖于 LibOS 网络服务功能本身的复杂程度，而且依赖于底层环境。目前，对于基于主机环境的 LibOS，如果主机 OS 已经实现了网络服务功能，则 LibOS 可以直接调用主机 OS 的网络服务功能接口实现 LibOS 的网络服务功能。但是，由于在主机 OS 中实现网络服务功能与传统宏内核一样会限制应用程序级 LibOS 的灵活性和高效性，因此当前研发的基于主机环境的 LibOS，如 ExOS 和 Graphene 等，均不在主机环境中实现网络服务功能，而是在 LibOS 中以应用程序的方式实现。例如在 Xok/ExOS 中，ExOS 作为 exokernel 系统的主要功能，已经实现了应用程序级的网路服务功能，这些服务包括 UDP/IP、TCP/IP、POSIX 套接字、ARP、DNS 和 tcpdump。Ganger 和 Kaashoek 等发现这种专门的应用程序要比基于套接字的版本快 8 倍[10]。

基于虚拟环境的 LibOS 多数都提供了网络服务功能。例如，IncludeOS 就已实现了必要的网络服务功能模块，如图 5 所示，其网络服务功能组件包括：Ethernet、ARP、IPv4、UPD、ICMP 协议库，而 TCP 和 IPv6 还正在开发完善中。另外，IncludeOS 还默认加载了网卡驱动，即 Virtio Net Device 驱动程序，其优点是 Hypervisor 不需要再模拟网卡设备，而是可以将网卡设备相关数据直接插入到由客户虚拟机共享内存中的队列里，从而提高运行效率。实验得到的测试结果显示，与在 Linux 上运行相同的网络服务功能二进制文件相比，IncludeOS 可以节省 80%～95%的 CPU 时间[25]。

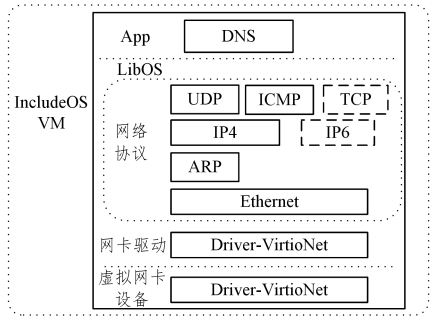

图 5　IncludeOS 中的网络服务模块

Fig. 5　Network service module in IncludeOS

4.2.2　文件 I/O 功能

文件 I/O 也是主机环境或虚拟环境中使用得最为频繁的功能之一，一般包括文件系统功能和磁盘驱动功能。无论是主机环境还是虚拟环境，磁盘文件 I/O 通常都是影响用户程序性能最大的瓶颈。传统访问磁盘文件的方法是通过文件系统接口，用户程序需要从用户空间转换到内核空间，通过磁盘驱动器访问磁盘上的文件，该过程不仅复杂，而且需要进行频繁的上下文切换，严重影响了磁盘文件的 I/O 性能。为了获得磁盘文件 I/O 的高性能，可通过 LibOS 来实现磁盘文件的 I/O 功能。当在 LibOS 中实现磁盘文件的 I/O 功能时，文件系统功能和磁盘驱动功能均在应用空间实现，应用程序调用文件系统 API，不需要运行环境的上下文切换，从而提高了磁盘文件 I/O 的速度。

当然，不同类型的 LibOS，其文件 I/O 功能的实现方式和复杂程度不仅依赖于 LibOS 文件 I/O 功能本身的复杂程度，而且依赖于底层环境。例如，文献[7]在主机环境下基于外内核操作系统 JOS 设计并实现了文件库操作系统。在实现该文件库操作系统时，文献[7]对 JOS 的文件系统功能进行了修改，重新定义了库操作系统的文件的基本数据结构，封装了磁盘驱动程序，将磁盘扇区抽象为磁盘块供文件系统进程访问，并用文件系统进程实现 POSIX 文件操作接口，体系结构如图 6 所示。依据外内核文件系统的设计原则，将文件系统进程运行在特权用户下，统一对磁盘空间进行分配和回收，非特权用户对磁盘进行操作时依靠文件服务程序的 IPC 通信；对于特殊的应用，用户可以通过此 LibOS 对磁盘直接进行操作。显然，实现基于主机环境的 LibOS 的文件 I/O 功能是相对简单的。

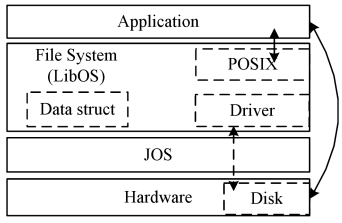

图 6 基于 JOS 的文件系统

Fig. 6 File system based on JOS

相反,实现基于虚拟环境的 LibOS 的文件 I/O 功能一般要比基于主机环境的 LibOS 的文件 I/O 功能复杂。例如,rump kernel 的文件 I/O 功能的实现就比文献[7]复杂,如图 7 所示。rump kernel 的文件 I/O 功能由 Fs-utils[49] 实现,Fs-utils[1)] 是一套用户空间文件系统实用程序,其中包含文件系统驱动程序 rumpfs[50]。rumpfs 是一个内存文件系统,在源模块 sys/rump/librump/rumpvfs/rump-fs.c 中实现,其特点是尽可能轻便和高效。大多数 rumpfs 操作只有简单的实现,并且高级功能的支持已经被省略,如重命名和 NFS 导出。

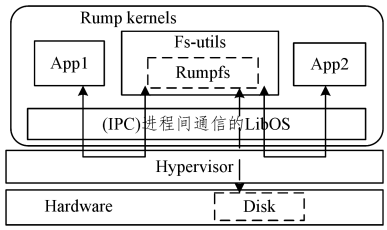

图 7 rump kernel 的文件系统

Fig. 7 File system of rump kernel

4.2.3 设备访问功能

除了网络服务功能和文件 I/O 功能外,设备访问功能也是主机环境或虚拟环境常用的功能之一,如键盘、软驱、U 盘、光盘、PCI 等。无论是主机环境还是虚拟环境,均需要提高设备访问功能的性能。传统设备访问的方法是通过设备接口,用户程序需要从用户空间转换到内核空间。通过设备驱动器访问该设备,该过程不仅复杂,而且需要进行上下文切换,严重影响了设备 I/O 的性能。为了获得设备 I/O 的高性能,可通过 LibOS 来实现设备访问功能。当在 LibOS 中实现设备访问功能时,应用程序调用设备 API,不需要运行环境上下文切换,从而提高了设备 I/O 的速度。

与网络服务功能和文件 I/O 功能一样,不同类型的 LibOS,其设备访问功能的实现方式和复杂程度不仅依赖于 LibOS 设备访问功能本身的复杂程度,还依赖于底层环境。目前,对于基于主机环境的 LibOS,如果主机 OS 已经实现了某设备的访问功能,则 LibOS 可以通过系统调用方式实现 LibOS 对此设备的访问功能,例如 ExOS 就按此方式实现了 Console 和 Keyboard 等设备功能。而对于基于虚拟环境的 LibOS,实现设备访问功能需要依赖底层 Hypervisor 的设备访问机制。例如,在 Xen 虚拟化环境中,MigrageOS 的设备访问功能由 MigrageOS 中的前端驱动程序和 Dom0 中的后端驱动程序组成,通过后端驱动程序将前端请求复制到一个真正的物理设备上来实现。图 8 展示了 Mirage LibOS 中某个设备被访问的过程。

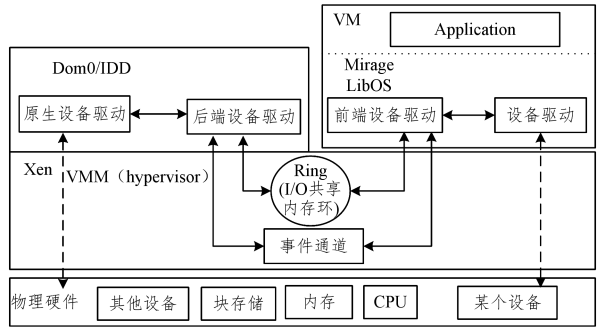

图 8 MirageOS 中设备访问功能的示意图

Fig. 8 Diagram of device access function of MirageOS

5 库操作系统面临的问题和挑战

目前,尽管已有 10 多个库操作系统项目正处于研发阶段,有的已开始局部应用,但是 LibOS 依然面临着各种问题和挑战。

(1)研发库操作系统的代价高。库操作系统通常是根据某类应用的特殊性能需求,由某一高级编程语言实现原本属于操作系统内核的某些资源管理功能,这就要求在研发库操作系统时,不仅要考虑具体的应用需求和编程语言的特点,还要考虑与底层操作系统之间的关联与界限,如果涉及新的硬件,还需要编写针对此硬件的驱动程序,非常复杂。即使底层操作系统内核已经实现了库操作系统所需的功能,从其内核中分离和提取这一部分代码也很困难,因为涉及到操作系统的方方面面,而且库操作系统必须与底层操作系统协同才能完成线程管理、CPU 调度和虚拟内存管理等基本功能。因此,研发一个库操作系统的复杂程度不低于同规模的操作系统。如果这类应用的特殊性能需求不强烈,据此研发库操作系统来提高应用性能会得不偿失。

(2)在扩展性和高性能方面,库操作系统存在两难。为了扩大库操作系统的应用范围,需要扩展库操作系统的适应性,包括多运行环境、多功能系以及多种编程语言的应用等。但扩展性的增加会提高库操作系统内核基的复杂性,降低库操作系统的性能。例如,为了扩展功能系,库操作系统不仅要增加功能,而且在内核基中也要增加新的代码来适应此新功能,这就会使得内核基越来越大,复杂性越来越高,与底层操作系统的内核越来越难以协调,顶层应用想要通过此库操作系统来获得高性能变得越来越困难。再如,将专一性的 LibOS 扩展为兼容性的 LibOS,涉及到针对不同的目标平台开发平台适配层,需要开发专门的交叉编译工具,另外还需要处理多种语言运行时之间的互操作难题,从而导致难以兼顾扩展性和高性能。

(3)库操作系统以及基于库操作系统的 Unikernel 的安全性需要加强。尽管 Unikernel 的安全性是通过底层操作系统或 Hypervisor 提供的隔离性来保证的,这比 Docker 仅通过 namespace 提供的隔离性安全得多,但库操作系统以及 Uni-

1) http://repo.rumpkernel.org/fs-utils

kernel 均是应用空间的一个普通进程，没有受到操作系统或 Hypervisor 的内核保护，容易受到多种传统攻击手段的攻击，如代码篡改攻击、缓冲区溢出攻击等。

同时，库操作系统还有多项关键技术有待优化和突破。

(1)LibOS 如何高性能地支持多进程应用程序

现阶段的大多数 LibOS 项目都是针对单进程应用服务的研究，但 Linux 的许多服务是多进程的，例如 Web 服务器和 shell 编译器，而对实际应用中的多进程应用程序的研究还存在诸多需要克服的难题，例如多进程之间的数据共享、通信和安全隔离性问题，资源合理分配(特别是内存开销)问题以及系统的性能问题等。因此，Graphene[29] 项目就针对该问题进行了有价值的研究和探索，能够通过支持英特尔 SGX(软件保护扩展技术)，让 Graphene unikernels 可以在硬件加密内存区域中保护关键应用程序，从而使应用程序免受恶意系统堆栈的影响，同时减少移植工作量。Graphene 值得后来的研究者们参考和学习。

(2)如何对在宏内核上开发的 LibOS 进行性能优化

性能优化主要是对宏内核结构在按照功能(包括线程管理、CPU 调度、虚拟内存管理和 I/O 设备管理、文件管理和网络服务功能等)进行合理划分后，能够根据实际应用程序的需求对相应的库函数进行专门的改写或重新开发，在编译时与应用程序进行链接，从而通过系统结构的优化实现良好的性能。

(3)如何处理多种语言运行时之间的互操作

对于具有兼容性的 LibOS 系统来说，开发应用程序的语言和实现 library(库)的语言可以不同。因此，要能够支持多种不同语言编写的应用程序，就涉及到对多种语言运行时之间的互操作进行处理的问题。

(4)如何优化基于虚拟化平台的库操作系统

基于虚拟化平台的库操作系统依赖于 Unikernel 和 Hypervisor 之间的上下文切换，而且 Unikernel 需要从 Hypervisor 的虚拟设备上传输数据，这可能会再次引入系统性能开销。

结束语　库操作系统与虚拟化技术的结合可以在云计算时代充分发挥出轻量级虚拟机的优势，促进大规模的虚拟化服务器应用以微服务的形式运行，从而节省宝贵的资源并提高运行效率和安全性。本文首先总结出了库操作系统的基本定义和基本特点，然后提出了库操作系统的分类模型，接着总结出了库操作系统的体系结构并详细阐述了库操作系统的关键技术，使得读者对库操作系统有一个全面的了解，促进库操作系统在我国的研发和应用。

现阶段出现的库操作系统各有特点，在不同的应用场景中能够发挥出各自独特的优势。但是，截止到目前，通过 LibOS 构建出的 Unikernel 仍有待研究，在实际生产配置中还面临着一些挑战。即便如此，LibOS 依然值得我们关注，因为对 Unikernel 进行研究很有价值，在今后的云计算环境和物联网时代其有望成为市场的一流产品。

参 考 文 献

[1] STALLINGS W. Operating Systems: Internals and Design Principles, Global Edition[J]. Bbc Video Library, 2014, 18(2): 235-244.

[2] MCBREWSTER J, MILLER F P, VANDOME A F. Kernel-computing: Monolithic kernel, Microkernel, Hybrid kernel, Exokernel, History of operating systems, Time-sharing, Unix, History of Mac OS, AmigaOS, History of Microsoft Windows[J]. Pesquisa AgropecuáriaBrasileira, 2012, 47(3): 336-343.

[3] MIAO H. Analysis of Practicality and Performance Evaluation for Monolithic Kernel and Micro-Kernel Operating Systems[J]. International Journal of Engineering, 2011, 5(4): 277-291.

[4] VALÉRIO C H. Microkernel Development for Embedded Systems[J]. Journal of Software Engineering & Applications, 2013, 6(1): 20-28.

[5] KLEIN G, ANDRONICK J, ELPHINSTONE K, et al. Comprehensive Formal Verification of an OS Microkernel[J]. Acm Transactions on Computer Systems, 2014, 32(1): 136-156.

[6] QIAN Z J, LIU Y J, YAO Y F, et al. Research on Method of Formal Design and Verification of Memory Management Based on Microkernel Architecture[J]. Acta Electronic Journal, 2017, 45(1): 251-256. (in Chinese)
钱振江，刘永俊，姚宇峰，等. 微内核架构内存管理的形式化设计和验证方法研究[J]. 电子学报，2017，45(1)：251-256.

[7] LI Z Z. A study of File System Based on Exokernel Operating System [D]. Lanzhou: Lanzhou University, 2016. (in Chinese)
李肇中. 基于外内核操作系统的文件系统研究[D]. 兰州：兰州大学，2016.

[8] ENGLER D R, KAASHOEK M F, O'TOOLE J. Exokernel: an operating system architecture for application-level resource management[J]. Acm Sigops Operating Systems Review, 1995, 29(5): 251-266.

[9] PULIDO B M H. Decentralizing UNIX abstractions in the Exokernel architecture[D]. Cambridge: Massachusetts Institute of Technology, 1997.

[10] GANGER G R, ENGLER D R, KAASHOEK M F, et al. Fast and flexible application-level networking on exokernel systems [J]. Acm Transactions on Computer Systems, 2002, 20(1): 49-83.

[11] LESLIE I M, MCAULEY D, BLACK R, et al. The design and implementation of an operating system to support distributed multimedia applications[J]. IEEE Journal on Selected Areas in Communications, 1996, 14(7): 1280-1297.

[12] HU J. Preliminary Research on Time Subsytem in Exokernel [D]. Lanzhou: Lanzhou University, 2014. (in Chinese)
胡俊. 外内核时间子系统的初步研究[D]. 兰州：兰州大学，2014.

[13] DONG W, CHEN C, LIU X, et al. SenSpire OS: A Predictable, Flexible, and Efficient Operating System for Wireless Sensor Networks[J]. IEEE Transactions on Computers, 2011, 60(12): 1788-1801.

[14] DENG L, ZENG Q K. Method to Efficiently Protect Applications from Untrusted OS Kernel [J]. Journal of Software, 2016, 27(5): 1309-1324. (in Chinese)
邓良，曾庆凯. 一种在不可信操作系统内核中高效保护应用程序的方法[J]. 软件学报，2016，27(5)：1309-1324.

[15] CRISWELL J, DAUTENHAHN N, ADVE V. KCoFI: Complete

Control-Flow Integrity for Commodity Operating System Kernels[C]// Security and Privacy. IEEE,2014:292-307.

[16] REUTHER A,MICHALEAS P,PROUT A,et al. HPC-VMs: Virtual machines in high performance computing systems[C]// High PERFORMANCE Extreme Computing. IEEE,2013:1-6.

[17] AMMONS G,APPAVOO J,BUTRICO M,et al. Libra:a library operating system for a jvm in a virtualized execution environment[C]// International Conference on Virtual Execution Environments (VEE 2007). San Diego, California, USA, DBLP, 2007:44-54.

[18] BAUMANN A,LEE D,FONSECA P,et al. Composing OS extensions safely and efficiently with Bascule[C]// Proceedings of the 8th ACM European Conference on Computer Systems. 2013:239-252.

[19] HERDER J N. Towards a True Microkernel Operating System. A revision of MINIX that brings quality enhancements and strongly reduces the kernel in size by moving device drivers to user-space[D]. Amsterdam: Vrije Universiteit Amsterdan, 2005.

[20] WU Z X. Advances on virtualization technology of cloud computing[J]. Computer Application,2017,37(4):915-923. (in Chinese)
武志学.云计算虚拟化技术的发展与趋势[J].计算机应用,2017,37(4):915-923.

[21] HE W,YAN G,XU L D. Developing Vehicular Data Cloud Services in the IoT Environment[J]. IEEE Transactions on Industrial Informatics,2015,10(2):1587-1595.

[22] KUMARA M A A,JAIDHAR C D. Hypervisor and virtual machine dependent Intrusion Detection and Prevention System for virtualized cloud environment[C]// International Conference on Telematics and Future Generation Networks. IEEE, 2015: 28-33.

[23] BARHAM P,DRAGOVIC B,FRASER K,et al. Xen and the art of virtualization[C]// Nineteenth ACM Symposium on Operating Systems Principles. ACM,2003:164-177.

[24] BALALAIE A,HEYDARNOORI A,JAMSHIDI P. Microservices Architecture Enables DevOps:Migration to a Cloud-Native Architecture[J]. IEEE Software,2016,33(3):42-52.

[25] BRATTERUD A,WALLA A A,HAUGERUD H,et al. IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services[C]// International Conference on Cloud Computing Technology and Science. IEEE,2015:250-257.

[26] MADHAVAPEDDY A,MORTIER R,ROTSOS C,et al. Unikernels:library operating systems for the cloud[C]// Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM,2013:461-472.

[27] MADHAVAPEDDY A,MORTIER R,ROTSOS C,et al. Unikernels[J]. Acm Sigplan Notices,2013,48(4):461.

[28] MADHAVAPEDDY A,SCOTT D J. Unikernels:Rise of the Virtual Library Operating System[J]. Queue,2013,11(11):30.

[29] Unikernels-Rethinking Cloud Infrastructure [EB/OL]. [2017-09-08]. http://unikernel.org/.

[30] Wikipedia. Unikernel[EB/OL]. (2017-08-12) [2017-09-08]. https://en.wikipedia.org/wiki/Unikernel.

[31] DUNCAN B,HAPPE A,BRATTERUD A. Enterprise IoT Security and Scalability:How Unikernels can Improve the Status Quo[C]// Sd3c 16 at the IEEE/ACM,International Conference on Utility and Cloud Computing. ACM,2016.

[32] MADHAVAPEDDY A,LEONARD T,SKJEGSTAD M,et al. Jitsu:Just-In-Time Summoning of Unikernels[J]. Intelligent Decision Technologies,2015,4(1):39-50.

[33] PORTER D E,BOYD-WICKIZER S,HOWELL J,et al. Rethinking the library OS from the top down[C]// Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM,2011:291-304.

[34] Microsoft Research. Drawbridge [EB/OL]. (2011-09-19) [2017-09-08]. https://www.microsoft.com/en-us/research/project/drawbridge/.

[35] hioa-cs/IncludeOS: A minimal, resource efficient unikernel for cloud services[EB/OL]. [2017-09-08]. https://github.com/hioa-cs/IncludeOS.

[36] MirageOS [EB/OL]. [2017-09-08]. https://mirage.io.

[37] Galois Inc. The Haskell Lightweight Virtual Machine (HaLVM) source archive [EB/OL]. [2017-09-08]. http://galois.com/project/HaLVM/.

[38] TSAI C C,ARORA K S,BANDI N,et al. Cooperation and security isolation of library OSes for multi-process applications[C]// Eurosys'14 Proceedings of the Ninth European Conference on Computer Systems. ACM,2014:1-14.

[39] Github. Graphene [EB/OL]. [2017-09-08]. https://github.com/oscarlab/graphene/wiki.

[40] MARTINS J,AHMED M,RAICIU C,et al. ClickOS and the art of network function virtualization[C]// Usenix Conference on Networked Systems Design and Implementation. USENIX Association,2014:459-473.

[41] BALLESTEROS F J. TD Lsub[EB/OL]. [2017-09-08]. http://www.lsub.org/export/clivesys.pdf.

[42] Erlang on Xen:at the heart of super-elastic clouds [EB/OL]. [2017-09-08]. http://erlangonxen.org/.

[43] LANKES S,PICKARTZ S,BREITBART J. HermitCore:A Unikernel for Extreme Scale Computing[C]// International Workshop on Runtime and Operating Systems for Supercomputers. 2016:4.

[44] LANKES S,PICKARTZ S,KREBS D,et al. HermitCore [EB/OL]. [2017-09-08]. http://www.hermitcore.org/.

[45] runtime.js - JavaScript library operating system for the cloud [EB/OL]. [2017-09-08]. http://runtimejs.org/.

[46] KIVITY A,LAOR D,COSTA G,et al. Osv—optimizing the operating system for virtual machines[C]// Usenix Conference on Usenix Technical Conference. USENIX Association. 2014: 61-72.

[47] OSv—the operating system designed for the cloud [EB/OL]. [2017-09-08]. http://osv.io/.

[48] GitHub. rumpkernel/rumprun [EB/OL]. [2017-09-08]. https://github.com/rumpkernel/rumprun.

[49] KANTEE A. Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels[D]. Finland:Aalto University,2012.

[50] KANTEE A. puffs-Pass-to-Userspace Framework File System [C]// Asiabsdcon. Tokyo:AsiaBSDCom Press,2007:29-42.