Unikernel 调研报告

ZOS 小组：冉礼豪，胡煜霄，戈惊宇，任正行，管修贤

2018 年 7 月 4 日

Contents

Chapter 1

云计算领域面临的挑战

1.1 引言

云计算的出现是互联网的传输速度，计算机计算能力的增强和个人、企业等计算机用户对计算资源的需求增加的必然结果，用户的数据，应用程序等在云计算中心托管，大大节省了用户自己的维护成本，然而这也给云计算平台提出了诸多挑战：

- 隔离：根据虚拟机的思想，我们希望呈现给用户的是一个整体的计算资源，因此一方面要实现用户之间的隔离，另一方面实现用户与操作系统的隔离，保证用户不会造成破坏。
- 性能：性能意味着托管的成本，执行相同量的任务，性能好的系统能带来更少的额外开销，高性能的云计算平台自然在一定的开销下能给用户提供更优质的服务。
- 安全：用户的数据，系统的运行状态等一旦被骇客入侵，将造成极大的损失。无奈目前安全问题是无法一劳永逸地解决的。

1.2 隔离

在个人用户的操作系统中，隔离已经十分重要，用户可能会运行很多应用程序，应用程序之间必须隔离。做到**“一个应用程序的崩溃不能影响其他应用程序，更不能影响操作系统。”**因此我们必须引入某种隔离机制，保证应用程序之间，应用程序和系统的隔离。在内存上，我们引入了用户空间和内核空间的

分离，在 CPU 的运行上我们设置了不同的运行级别，一些敏感的操作被设定在更高级别的运行状态下，这样能阻止本不应该运行的指令破坏系统。

而在云计算平台上，这样的隔离变得更加复杂，主要体现在虚拟机上。首先，我们无法保证用户托管的应用是安全的，也无法保证用户做出安全的操作，因此我们引入更强的隔离机制——虚拟机。用半虚拟化或者完全虚拟化的方式模拟一台物理机器，用户的应用程序和操作在上面的运行同一台真实机器上的效果完全等价，这样用户可以随心所欲地在上面执行各种操作而不会破坏宿主操作系统，在理想情况下（没有严重漏洞）也不能访问其他的虚拟机空间。其次，云计算平台提供给用户的应该是一个完整的，满足用户需求的运算服务，这意味着必须提供一个和一台真实主机效果相同的运算环境，为了减少资源分配所带来的麻烦，引入了虚拟化技术和容器技术，使用户提交的应用在一个可定制，有很强隔离性的虚拟环境中运行。

1.3 性能

虚拟化技术引入的隔离机制很好地保证了整个体系的运行和管理，但是无疑也加重了云计算平台的负担。这里需要说明软件栈的概念——指一个应用程序运行所依赖的环境，包括运行库，操作系统中的硬件驱动，进程管理，内存管理等模块。在传统的虚拟机模式下，我们的宿主机已经有了一个相当庞大的软件栈，而我们又在这基础上搭建虚拟化的操作系统，又一次增高了软件栈。这样的解决方案无疑是低效的。

这样的状况在云计算中心意味着什么呢？我们需要浪费大量的空间去执行非常简单的任务，执行这些任务之前的初始化占用的资源甚至比应用程序本身还要多。

为了改进性能，一种以 Docker 为代表的应用程序级的虚拟化技术迅速发展起来，docker 其实是一种将应用程序和运行库，软件依赖打包分发的技术，再结合镜像站等网络服务，为开发者提供了一个非常完整的开发链，因而迅速得到了普及。

还是由于软件栈的原因，每个应用程序都有它的运行环境，这些环境是由开发者使用的技术决定的，然而这并不是好事，因为开发者使用的环境和真实的运行环境很难做到完全一致。比如开发者编写的程序依赖的库，库的版本，操作系统等等都会和使用场景不一样，这样给开发和测试造成了很大的不方便。

综合上述面临的问题来看，很多问题都涉及到同一个点：操作系统太过复杂，使我们很难确保执行每一个功能的模块都是安全的，还会让操作系统的效率降低，还会使开发与部署变得非常繁琐。然而在云计算平台上，这样的缺点

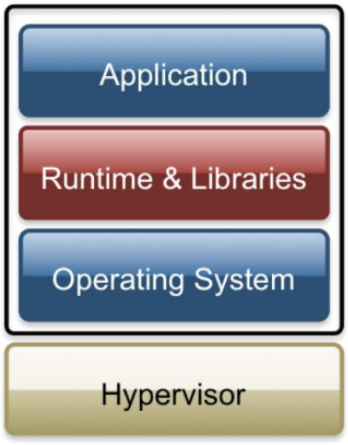


Figure 1.1: vm struct

有望通过容器技术得到解决。

1.4 安全性

无论是个人 PC 机，云计算平台还是嵌入式系统，安全性从来就没有一个一劳永逸的解决方案，也许这不仅是一个技术问题，也是一个哲学问题。但无论如何，现在操作系统的复杂程度决定了我们很难真正掌握一个系统的安全状况。最常见的模式是：发现一个漏洞，修复一个，以后再发现漏洞。现有操作系统的冗余复杂，使得对计算机系统的维护变的十分困难，并且留下了大量的漏洞和攻击面 (attack surface)，给恶意黑客以可乘之机。类似的例子有很多，拿 windows10 为例，仅 2016 年一年内，Microsoft 为其修补的漏洞就达 729 个之多。“WannaCry”病毒也是利用 windows 系统中的“Enternalblue”漏洞来进行进攻的。

尽管安全问题可以通过内核的可信化验证或者使用新的编程语言 –比如 RUST，得到解决，但是我们必须承认，功能越是复杂的系统，安全越成问题。

安全问题在云计算时代直接与大量的宝贵信息和计算资源挂钩，云计算中心最禁不起严重漏洞造成的风险。Amazon, Microsoft, Yahoo, Linkin 等公司都遭受过攻击，造成用户的信息泄漏，严重威胁用户的安全。2017 年 5 月，“WannaCry”勒索病毒席卷全球，数十万电脑用户中招，造成的直接经济损失超过 80 亿美元。同年 10 月，东欧银行网络遭到黑客入侵，损失达上亿

元。据不完全统计，每年全球仅因黑客攻击导致的直接经济损失达 4500 亿美元。从个人计算机到银行、政府的大型主机，无一不面临着黑客、木马等不安全因素的威胁。

Chapter 2

unikernel 技术的思想

2.1 容器化技术解决的问题和实现原理

容器化技术，这里以 docker 为例，给云计算平台上的应用程序开发带来了一场革命，docker 技术解决的问题就是软件的运行环境。Docker 是一个开放源代码软件项目，让应用程序布署在软件容器下的工作可以自动化进行，借此在 Linux 操作系统上，提供一个额外的软件抽象层，以及操作系统层虚拟化的自动管理机制。Docker 利用 Linux 核心中的资源分离机制，例如 cgroups，以及 Linux 核心命名空间（name space），来建立独立的软件容器（containers）。这可以在单一 Linux 实体下运作，避免启动一个虚拟机器造成的额外负担。

Docker 的成功是无疑的，它采用了容器化隔离的方式，容器之间需要共享一个操作系统，也分享操作系统提供的服务，这与传统的虚拟机不同。传统的虚拟机真的模拟出一个与宿主机独立的环境，拥有自己的软件和库，容器化技术相当于解决了不同虚拟机各自拥有了一套独立而重复的运行环境的问题。

除了更少的空间占用外，相比传统虚拟机，docker 运行的效率更高，这是因为它并不是一个完整的虚拟机，Docker 直接运行在 host 上，而不用在一个虚拟化出来的相对较慢的计算机上运行。并且它也不需要像传统虚拟机一样等待系统 boot，像一个进程一样被 host 管理。

综上所述，相比传统虚拟机，docker 在资源占用，运行速度和效率上都有一定的优势，但是它并没有解决安全问题，这不是 docker 的设计初衷。

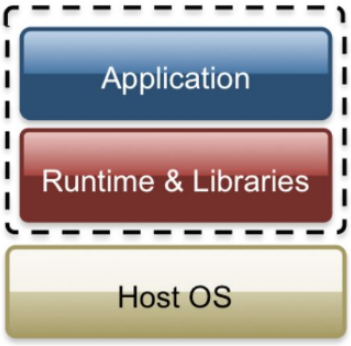


Figure 2.1: container

2.2 Unikernel 解决的问题和实现原理

相比以 docker 为代表的容器技术，Unikernel 在追求性能和精简方面做的更好，它确实比 docker 做得更加极端：不仅打包了程序运行所依赖的库，还有它需要的系统服务，也就是说，它完全作为一个专门化的虚拟机镜像运行，而不需要任何多余的模块。

Unikernel 为了实现体量的缩小，选择了与 Docker 完全不同的路线。它只专注于一个目标程序，没有其它冗余的程序和支持，没有多进程切换，所以系统很小也很简单。虽然 Docker 容器的 image 比传统的虚拟机 (以 G 计) 要小很多，但是一般也是好几百兆，而 unikernel 由于不包含其它不必要的程序和服务 (ls,echo,cd,tar 等)，甚至没有 Shell，所以体积非常小，通常只有几兆甚至可以更小。比如 mirageOS 的示例 mirage-skeleton 编译出来的 xen 虚拟机只有 2M。体量的缩小也使得 Unikernel 的启动和运行的速度相较 Docker 有明显的提高。

另外，Unikernel 没有其它不必要的程序，使其受到漏洞攻击的可能性大大降低，docker 里面虽然大多数情况只跑一个程序，但是里面含有其它 coreutils，只要其中任何一个有安全漏洞，整个服务可能就会受到攻击。而 Unikernel 中没有 Shell 可用，没有密码文件，没有多余的设备驱动，只有少数功能可以利用，这严重限制了未经授权的用户可能造成的破坏。另一方面，对于 Unikernel 而言，每一个操作系统都是定制用途的，其中包含的核心库均不相同，即使某个服务构建的操作系统由于特定软件问题而遭到入侵，同样的入侵手段在其他的服务中往往不能生效。这无形中增加了攻击者利用系统漏洞的成本。

这样的设计依赖于“库操作系统”，这并不是一个真正的操作系统，U-

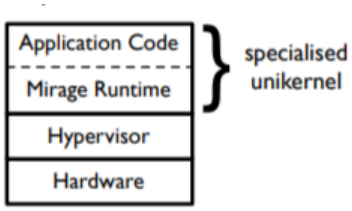


Figure 2.2: unikernel

nikernel 的开发过程与嵌入式系统很像，类似于”交叉编译“，对于编好的程序，我们需要找到它运行所需要的最少条件，从库操作系统中选择出来与程序打包成虚拟机镜像。这意味着它很可能不能运行其他的程序。

Unikernel System，一个知名 Unikernel 开发研究组织在 2016 年一月被 Docker 收购。作为当下最流行的应用容器引擎服务的提供商，Docker 这一收购足以说明 Docker 对于 Unikernel 的重视。

在传统的操作系统当中，内存被分为内核空间和用户空间，内核空间有操作系统提供的服务和运行库，包括一些底层的硬件驱动，文件系统，内存管理等等。这样的设计也同时提供了用户与系统的隔离，进程的调度，以及其他多用户操作系统需要的功能。用户空间则提供具体的应用程序功能，在计算机的使用者看来，用户空间包含的是要运行的指令，而内核空间为用户空间的代码提供具体的功能实现的服务。

一个传统的应用程序的运行需要依赖操作系统和运行库，依赖系统调用和各种 I/O 操作的支持才能正常完成其功能，这样的过程往往需要很多步调用和寻址，造成了很大开销，如果能让应用程序自己有驱动，直接使用硬件资源，效率将大大提高。

这样的精简就是 unikernel 的主要思想，unikernel 中，软件栈变得不一样了：

可以看到这里已经没有用户空间和内核空间之分了，传统的设计是内核，运行库，应用程序。而在 Unikernel 中所有都是一体的，镜像中也只有一个应用程序运行，这里的代码已经包含了完成工作需要的所有代码。只需要作为一个系统镜像启动，完成所有的功能。

看起来这种设计理念貌似是非常原始的，然而从操作系统的设计上出发我们会发现，最早人们设计系统调用，就是为了节省程序员在编写底层驱动上消耗的时间，专注于软件功能本身。实现这个目的的方式不仅有是设计一个“包罗万象”的操作系统，我们还可以根据运行程序的需要，为每个应用程序“定制”操作系统，当然这个定制过程是自动化的。构建 unikernel 的过程很像交

叉编译，即在本地机器上编译一个针对其他平台的程序，只不过这里所谓的其他平台是一个已经移除了对于该应用来说不必要的内容的精简环境，可以将该环境理解为 hypervisor。即虚拟机管理系统。

在实际使用过程中，因为我们需要充分利用硬件资源，运行很多 unikernel，在这种情况下，只需要为每个应用程序都制作 Unikernel，再利用 hyper visor 对它们进行统一的管理即可。

2.3 Unikernel 与 docker 的比较

Unikernel，Docker 二者经常被拿来相提并论。它们的主要意义都是用来替换云计算中使用传统虚拟机这一过于臃肿和有时不太安全的解决方案，但两者的区别非常明显。

Docker 为云计算的操作系统提供了相较传统虚拟机一个更轻量级的选择。在很多方面，Docker 下的容器与虚拟机非常相似。它们都旨在为运行代码提供一个能支持代码运行的独立稳定的环境。最大的区别是 Docker 通过分享来减少重复。Docker 可以共享主机环境的 Linux 内核，也可以共享操作系统的其余部分，甚至它们还可以共享除应用程序代码和数据之外的所有内容。例如，我可以使用容器在同一台物理机器上运行两个博客。这两个容器可以设置为共享除模板文件，媒体上传和数据库之外的所有内容。借助一些复杂的文件系统技巧，每个容器都可以“认为”它具有专用的文件系统。

Chapter 3

Unikernel 的优势和劣势

3.1 优势

Unikernel 采用的设计专门操作系统的想法并不是第一个，eliminate general purpose operating system 的想法早就有人提出来了，但是应用的场景实在太小，无法得到大范围应用。但是现在 Unikernel 的适用环境恰恰符合云计算平台 –我们需要让虚拟机更小，更快，更安全。

小：很多虚拟机镜像因为移除了复杂的软件栈，常常不足 1MB，完成的功能相同，这并不奇怪 –比如我要写一个 hello world 程序，不需要 shell，不需要网卡驱动，只需要一些 I/O 的管理和进程调度即可。当前的软件堆栈（software stack）一般由成百上千个部分组成，经常需要使用千兆字节的内存和磁盘空间。他们花费大量宝贵的时间用于启动和关闭，大而缓慢。自从 IBM PC 出现之后，人们构建软件堆栈的方法几乎没有变化。37 年来，我们一直在采用在硬件大而缓慢的时候设计的堆栈方法。如今，硬件系统已经发生了天翻地覆的变化，操作性能比过去有了数千倍的提高，为什么我们还在使用这种已经显然过时的堆栈方法呢？如果硬件系统发生了如此大的变化，那么软件系统是否也需要相应的进化以适应硬件的发展呢？

答案是显然的。Unikernal 的出现正是对这一需求的呼应。想象一下一下子抹去 95% 的内核（你的应用根本不需要那些）是一种什么体验？省出大量的硬盘空间、应用运行得更加流畅——Unikernal 正是这样！Unikernel 镜像都很小，由 MirageOS 实现的一个 DNS server 才 184KB，一个 w eb server 仅仅 674 KB，小到令人恐怖的程度。因此开发在 Unikernal 上的项目通常没有传统操作系统的弊端，这无疑使它更加适应现代硬件系统。

快：移除了传统的软件栈设计之后，Unikernel 的风格是直入主题，将一切变得简单化，效率自然变得很高。对传统操作系统来说，也许那十几秒的开机时间在你看来并不慢，但是在大型主机中，现有计算机的运行速度仍不能很好地满足要求，还有极大的改进空间。这一改进可以从计算机硬件入手，比如制造更大规模的集成芯片、提高 cpu 主频等等，但如果从操作系统入手，可能会产生更好的效果。

Unikrenal 可以说是从操作系统入手，提高电脑运行速度的一个很好的尝试。由于 Unikernal 的轻量，精简了冗余的结构，删减了多余的步骤，其启动与停止都在毫秒级别，远远快于常规操作系统。运行在其上的应用程序也更加高效。

安全性：这可能是 Unikernel 的最重大的优势，其实很多漏洞与应用程序没有关系，反而与应用程序并没有调用的系统服务有关系，移除了不需要的内容，也就大大降低了系统被攻击的机会。举例来说：设计了一个与用户进行通信的网络服务器，如果用户要攻击骗取控制权，它会发现很难下手，因为除了服务器需要的功能，其他的部件都被移除了，甚至连 shell 都没有。Unikernel 的设计在保证了程序正常运行的同时，也大大限制了攻击者。在计算机领域，有时候小而简单的东西相对而言更加安全，维护起来也更加容易，Unikernal 就是这样。Unikernal 中没有 shell，没有密码文件，也舍去了那些可能成为干扰源的视频和驱动。这极大地降低了 Unikernal 发生错误的可能，带来了更少的漏洞和更小的攻击面，大大地提高了安全性。

3.2 劣势

然而 Unikernel 并不是现代操作系统的救星，它也需要在一些优势和劣势之间作出权衡。其中最明显的几个劣势如下：

单进程：Unikernel 的内部环境更像一个裸机，线性的，原始的内存状态，只有一个进程（不过多线程还是可以实现的），其实要做一个多进程的程序也不是不可能，但是进程管理（启动，终止，监视进程等等）会造成很大的开销，Unikernel 的优势会被削弱。

单用户：需要说明的是，单用户并不一定是缺陷，同样多用户的支持也需要很大的开销，如果要支持多用户，那就需要登录，验证，权限管理，用户之间的隔离，还要有一些安全验证防止欺骗。但问题是：为什么一定需要多用户呢？Unikernel 使用场景决定可以让一个用户占有一整个虚拟机。

糟糕的 debug：如果在开发过程，那样程序的 debug 比较简单：和正常的 debug 过程没有什么区别。但是在运行过程中如果出了问题，我们看不到栈，

没有 tcpdump，没有传统的 debug 信息支持，我们很难知道到底出了什么问题。

3.3 Unikernel 的应用场景

在了解了 Unikernel 的一些劣势之后，我们很容易得出什么时候该用 Unikernel，从而最大程度发挥 Unikernel 的优势.

- 不需要在一台机器上运行多个进程。
- 只需要单用户。
- 需要非常快的开启速度。
- 需要接入网络，对安全性有很高的要求。
- 可能需要大量部署的程序。
- 程序自带一套机制提供程序崩溃时的诊断。
- 不需要经常迭代版本或进行维护。

Chapter 4

Unikernel 的生态

一项技术的广泛使用需要开发者为其搭建方便开发、部署、管理的生态环境。目前为止，与 unikernel 开发，部署，运行的相关项目有：

4.1 Jitsu

JitSu 是“Just-in-time summoning of unikernels”的缩写，JitSu 利用了 unikernel 极快的启动速度，实现了服务的即时启动，平时空闲的状态，事实上它是一个 DNS 服务器，建立了服务器 (unikernel) 和请求之间的映射，当它收到请求时，会快速启动 Unikernel 来对请求者提供服务，这样相当于快速地生成了服务器，而不需要让服务器一直处于一个等待连接的状态一直占用资源，这样做提高了计算资源的利用效率。

Jitsu 的作用就在于它实现了一个叫临时微服务器的概念 (Transistent microserver)，因为 Unikernel 的 boot 时间只有几十毫秒，是很理想的动态建立服务器的方式。

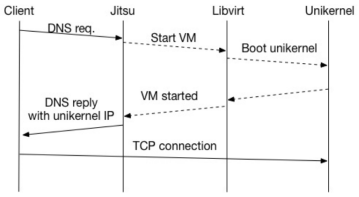


Figure 4.1: jitsu structure

4.2 Rump Kernels

Rump Kernel 项目极大地降低了 unikernel 的门槛。Rump Kernels 的设计目的是尽力实现在不修改应用的原有代码结构的情况下将其做成 unikernel，Rump kernel 基于 NetBSD, 一个跨平台性能很好的操作系统，为了实现很好的跨平台性，NetBSD 的设计非常适用于各种不同的硬件平台，替换某个硬件驱动后仍可以正常运转，所以 NetBSD 可以在很多硬件平台上部署。Rump Kernel 支持比较传统的 POSIX 标准，因此很多应用只需要做很少的修改就可以移植到 Rumprun Unikernel 上。

Rump Kernels 是 Rumprun（一个支持 POSIX 的 Unikernel 实现）的基础，因为在 Unikernel 的开发中，需要做的其实是一个交叉编译，目标平台与自己的机器并不相同。所以需要定制专门的编译器。

4.3 Xen Project Hypervisor

Unikernel 的本质是一台虚拟机，而 Xen 虚拟化技术提供了一种叫 paravirtualization 的技术。这项技术使得一些虚拟机“知道”自己在一个 H ypervisor 上而不是一台真正的机器上。那么虚拟机与 Hypervisor 的交互就不需要“模拟”一台真实的机器了。

这项技术使得 Xen 项目成为 Unikernel 的首选运行平台，paravirtualization 使得 Unikernel 可以非常高效地与宿主机进行通信，使得启动并管理大量的 Unikernel 成为可能，目前，在运算规模随 Unikernel 的数目成线性增长的情况下，hypervisor 可以同时运行大约 1000 个 Unikernel。

4.4 Clive OS

Clive 是一个云操作系统，旨在提供简单高效的云计算服务，目前主要被用来虚拟机上提供进一步的开发支持。

简单来说，Clive 是一个用 Go 语言设计的 Unikernel 实现，设计者修改了 Go 语言的编译器和 Runtime 来提供更好的网络服务。

4.5 MiniOS

MiniOS 是在 Xen 项目的基础上的操作系统，MiniOS 可以用来运行大量的 Unikernel 应用，包括 MiregeOS 和 ClickOS。而对于自己来说，MiniOS 什

么也不做，它的价值在于很容易被修改使 Unikernel 可在上面运行。它简化了 Unikernel 的开发周期。

4.6 UniK

UniK 是一个用 Go 语言编写的项目，目的是让开发者轻松地将应用编译打包成对应平台的 unikernel，包括主流的 Rumprun，OSv。以及各种硬件平台，包括物联网设备上运行的，UniK 项目提供了一系列 API 使得开发者不必再学习各种不同的 unikernel 的工具链，大大减小了开发代价。

Chapter 5

Unikernel 的改进可能

5.1 更进一步：Unikernel Monitor

现有的 unikernel 运行在某种监视程序上，比如 qemu。

也就是说，我们用 Unikernel 的方法专门化的部分是 OS，然而，其实在 Unikernel 下面一层的 Monitor 也可以进行 Specialize, 这样它们直接运行在 hypervisor 之上 (比如 xen 和 kvm)，这样做的好处当然是进一步优化了软件栈，可以获得更高的运行效率。

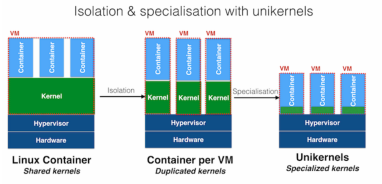
5.2 Debug 环境的优化

因为 Unikernel 砍掉了运行中不需要的内容，并且没有底层 OS 的支持，这决定它很难像普通的应用程序一样，通过一些进程管理和日志机制进行调试，但是这并不代表 Unikernel 是完全不可调试的，其实我们已经不能像看待传统的虚拟机一样看待 Unikernel，应该对 Unikernel 运行过程中出现的意外情况有相应的保护措施和追踪记录，以此方便开发者进行维护。这也是我们大作业想要着手改善的内容。

5.3 Unikernel 和 Docker 结合

Unikernel 和 Docker 的思想类似 –打包，但是二者各有优缺点和合适的使用场景，二者不是竞争关系，并不存在一个淘汰另一个之说。

2015 年的 Hyper 项目实现了在 hypervisor 上面运行一个 Docker，相当于



拿掉了操作系统一层，这样的技术就与 Unikernel 非常类似。这么做大大限制了攻击者的空间，提高了 Docker 的安全性。

Chapter 6

我们要做什么

由于 Unikernel 的结构组成的特殊性，一些程序想要移植到 Unikernel 上往往需要将代码完全重写（而且现在比较成熟的 Unikernel 实现往往需要使用冷门的编程语言，开发接口的难度也很大），这一定程度影响了可移植性。而且 Unikernel 并不是传统的程序，开发和测试都与传统的有不小的差别，造成了开发难度的增大。

在开发过程中，开发者可以在传统的操作系统上进行开发，而所有内核相关的功能，暂且由开发机的操作系统提供。

而在测试环境中，大部分 Unikernel 的实现会将应用代码与需要的内核模块构建成 Unikernel 后，再将其跑在一个传统的操作系统上，利用传统操作系统上的工具来测试 Unikernel。以 Rumprun 为例，它可以通过 KVM / QEMU 来运行一个 Rumprun Unikernel VM，随后用 Ho st OS 上的 GDB 来对其进行调试，这跟我们实验中测试内核的步骤比较相似，那么我们在实验中对于内核调试的种种遗憾，如调试器容易有 bug ，编译调试步骤繁琐，命令或者说功能不够丰富科学等等，也可以投影到 Unikernel 的开发调试上。联想诸如 Pycharm ，Vs Studio 这些成熟的 IDE，它们丰富的调试辅助功能，对开发效率的提高是非常显著的，也变相地降低了编程开发测试的难度，降低了编程的门槛，促进了语言的普及。那么，如果我们对于 Unikernel 的测试环境能有所改进，小到改进 debugger 一些缺点，大到开发一个相应的 IDE，就可能为促进 Unik ernel 的实用化和普及，创建一个更高效的云计算世界做出一份贡献。这正是我们想要做的。

Chapter 7

参考资料

An introduction to unikernels-Unikernel.org

Antti-The rise and fall of the operating systems

JitSu: just in time summoning of Unikernels

Anil Madhavapeddy, Richard Mortier1, Charalampos Rotsos, David Scott2, Balraj Singh, Thomas Gazagnaire3, Steven Smith, Steven Hand and Jon Crowcroft-Unikernels: Library Operating Systems for the Cloud

到底什么是 unikernel?

Unikernel-wikipedia

The Clive Operating System

rumpkernel-FAQ wiki

UniK Repository

Unikernels are unfit for production | Hacker News

Unikernels are secure. Here is why-10 July 2017 by Per Buer

Unikernels O'RELLY Media ISBN: 9781492042815

Unikernel Systems joins Docker-26 November 2015 by Martin Lucina