[x-unipanic]

1. **项目简介**：在已有项目的基础上，继续改进 Unikernel 的二进制兼容性和易用性，做出可以即使打包、分发的 Unikernel。Unikernel 是一个安全、轻量而高效的运行环境，其应用前景十分广泛，但存在的一个问题是，无法与未知源码的二进制程序打包，所以此项目希望在保持 Unikernel 现有优势的前提下，改善 Unikernel 对二进制程序的支持。致力于提供二进制兼容性的 Unikernel 项目 HermiTux 仍有较大改进空间，所以该项目将在 HermiTux 基础上，把改善 HermiTux 二进制兼容性作为立项目标。
2. **关键词**：Unikernel，二进制兼容性，在 HermiTux 和 Kylinx 基础上
3. **难度评价**：
   a) 在掌握操作系统有关用户与内核、进程调度等等基础概念的基础上，需要花费一定时间去学习 Unikernel，学习 HermiTux 和 Kylinx，学习难度高,寻找合适参考的难度也存在
   b) 该项目最终在 HermiTux 基础上做出了两点改进，使其支持多进程/线程，改写了改写系统调用的逻辑
4. **有价值的信息**：
   a) Docker：Docker 简介 - Docker — 从入门到实践 (gitbook.io) Docker 是一个开源的应用容器引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。属于操作系统层面的虚拟化技术。Docker 的出现，让应用环境的配置、发布和测试变得

异常轻松。Docker 在概念上与虚拟机极其类似，但却轻量很多，它为每一个应用提供完全隔离的运行环境，这个“环境”在 Docker 中也被称为 container 容器。Image 镜像，可以理解为虚拟机的快照，里面包含了你要部署的应用程序以及它所关联的所有库。容器就像是一台台运行起来的虚拟机，里面运行着应用程序。Dockerfile 就像是一个自动化脚本，主要被用来创建镜像。

b) Unikernel:Unikernel 是精简专属的库操作系统(LibraryOS)，它能够使用高级语言编译并直接运行在商用云平台虚拟机管理程序之上。Unikernel 使用户可以从一个服务库中选择应用需要的操作系统服务而无须整个操作系统，用户选择的服务则会成为应用的一部分。Unikernel 是容器技术发展的必然产物，小、简单、安全、高效，当你看到云客户端时就像看到单应用硬件一样，那就是 Unikernel 试图解决的：删除应用与硬件中间臃肿的部分，从而让最“精简”的操作系统运行你的代码。

c) 善于在前人的基础上进行学习、借鉴和改进。

[x-sBPF]

1. **项目简介**：该项目是一个轻量级的文件系统沙盒，通过劫持文件访问相关的系统调用，实现对文件的保护。项目的特色在于沙盒程序是一段动态植入内核空间的 kernel 代码，可以最小化反复切换特权级别带来的性能损失。用户态沙盒面临性能缺失和程序被篡改

的安全问题，用户态沙盒灵活性差，于是本项目采取了班用户态半内核态的沙盒，用户编写沙盒程序并直接注入内核，在内核中完成沙盒执行。

2. **关键词：**沙盒，文件保护，安全
3. **难度：**能看出本项目查阅了大量的参考资料，同时也对 Linux 许多代码进行了阅读分析。在对内核态进行修改时，需要对用户态内核态的概念有深入的理解。整体而言，难度较高。而且他们在实际实现过程中遇到过难以解决的困难，最后不得不修改设计思路。
4. **有用的信息：**
   a) 沙盒：在计算机安全领域中是一种安全机制，为运行中的程序提供的隔离环境。沙盒将软件运行于一个受限的系统环境中，控制程序可使用的资源（如文件描述符、内存、磁盘空间等）。
   b) BPF: BPF 提供了一种在不修改内核代码的情况下，可以灵活修改内核处理策略的方法。这在包过滤和系统 tracing 这种需要频繁修改规则的场合非常有用。因为如果只在用户态修改策略的话那么所有数据需要复制一份给用户态开销较大；如果在内核态修改策略的话需要修改内核代码重新编译内核，而且容易引入安全问题。BPF 这种内核代码注入技术的生存空间就是它可以在这两者间取得一个平衡。
   c) 他们的实验总结，我认为对我们的后续工作有启发意义，所以截图在此。

5.3 项目总结

本项目在整个过程中历经波折，甚至修改过立项之初的一些想法，最终才达到了今天的完成状态。期间本组的队员们遇到了很多超出想象外的困难，并且在解决这些困难的过程中增加了我们对 linux 内核运行过程的很多理解。开学时，我们对操作系统的运行原理一窍不通，甚至一开始时对内核态用户态的区别都一切不通。通过在内核态进行很多的修改和测试，我们解决很很多诸如内存管理权限问题、不同用户进程对内存地址的理解不同的问题（由于段页式内存对每个进程进行的内存抽象），在完成了此项目的过程中我们通过查找资料与多种编程尝试，成功的从理论上和实践上解决了这些问题。在项目的整个推进过程中，我们也实现了成功的分工和时间规划，高效有序的完成了整个项目的推进，并且最终完成了这个项目。

[x-chital]

1. **项目简介：**gVisor 是一种轻量级的容器技术,通过系统调用劫持，为应用程序打造一个虚拟的内核环境，gVisor 是基于进程虚拟化的容器实现，他拥有很好的隔离性，很小的内存占用和启动时间，但是系统调用效率不高。所以本项目 rVisor 为一个由 rust 编写的基于用户空间的通用安全沙箱环境。rVisor 利用在内核中进行系统调用劫持的方法，避免频繁的内核态用户态切换所带来的性能开销，同时使用 Rust 语言，在保证安全性的同时解决 Go 语言 GC 所带来的 Stop the World 问题，力求获得可以媲美原生应用的性能。实现了一个简易的拥有独立内核的安全沙箱架构。
2. **关键词：**容器，gVisor，性能，rust，zCore
3. **难度：**从容器出发，到为解决容器安全性问题的 gVisor，再到为解决 gVisor 性能的项目，思维逻辑较为清晰。本项目需要学习容器，rust，gVisor，不仅是了解功能，更是需要分析其实现逻辑。

借鉴了 zCore，作为本项目的内核。

4. 有用的信息：

a) gVisor 通过截获应用程序的系统调用，将应用程序和内核之间完全隔离。gVisor 没有简单的把应用程序发出的系统调用直接作用到内核，而是实现了大多数的系统调用，通过对系统调用模拟，让应用程序间接的访问到系统资源。gVisor 模拟系统调用本身时对操作系统执行系统调用

b) 容器化、微服务化、云原生化是当前 IT 系统演进的趋势

c) ptrace：ptrace 提供了一种机制使得父进程可以观察和控制子进程的执行过程，ptrace 还可以检查和修改子进程的可执行文件在内存中的 image 及子进程所使用的寄存器中的值。通常来说，主要用于实现对进程插入断点和跟踪子进程的系统调用。

d) zCore:zCore 是一个比较成熟的、轻量的、使用 rust 语言的操作系统。如果将来要用 rust 语言设计操作系统，可以作为参考

e) Kata Containers: 是一个可以使用容器镜像以超轻量级虚机的形式创建容器的运行时工具. 它给每个容器（在 Docker 容器的角度）或每个 Pod（k8s 的角度）增加了一个独立的 linux 内核（不共享宿主机的内核），使容器有更好的隔离性，安全性.基于 kata containers 创建的容器兼具虚机和容器的优点，在隔离性上，跟常规虚机类似，但在部署时间和运行效率上却

与常规容器相近.

f) 我们知道攻击者会利用 Linux 内部的漏洞来实施攻击，那么让 Linux 内核解决全部安全问题是否可行呢？事实上，这还是一件比较困难的一件事。

主要原因在于，Linux 他自身在不断更新和扩展，内核会不可避免地变得越来愈大，在这种扩大的过程中，安全是不可避免的问题。

Linus 针对这种安全性问题的回答是，一方面在对安全性要求不高的场合下，你可以允许这些 bug 发生，另一方面，在安全性要求比较高的场合下，可以在 Linux 的基础上对计算资源建立一层新的抽象。这样的话，如果一个组件有一个漏洞，另一个组件或许能够捕获到它，这样就可以更大程度上保证安全。

g) 一种源远流长的独立内核的方案是 unikernel，让应用自己带上自己的内核，这种方式的好处是最简化的 LibOS 不仅可以有更小的开销，也可以有更小的攻击面，但阻止它被更广泛应用的问题在于它往往需要修改应用，兼容性永远是平台技术被采纳的最大障碍。

[总结]

1. 我本次调研的三个项目中，有两个都是安全相关的方向，基本思想都是隔离，一个是尝试编写半用户半内核的沙盒程序，一

个是改进 gVisor 的性能问题。其中 x-chital 是我最欣赏的一个项目，它的思考逻辑清晰，文档详细明了，实现了很有价值的成果。

2. 受最后一个项目的影响，容器安全是我目前感兴趣的一个领域，我认为这一方面代表着未来。容器本身具有其引人注目的优势，而安全性又是影响容器大规模应用的一个重要因素。
3. 另一个我感兴趣的方向就是，在万物互联的趋势下，每个东西都是相互联系配合的，在这个过程中，操作系统能够在不同物品之前的通讯、同步，以及一些在本地不能做的计算放到云端上，这些方面能够发挥什么作用。