计算机系统（2）实践手册

作者：ACM 班课程助教
组织：ACM 班，上海交通大学
时间：2024-2025 春季学期
版本：1

目录

实践任务的背景

在 21 世纪 20 年代，机器学习的极具发展、大模型的快速迭代使得操作系统相关的内容成为“不受大家待见”的东西。学术界急速转向机器学习系统相关系统使得大家对于传统硬核的操作系统失去了兴趣。不仅班级内发生了转向，大多数实验室也在这一期间放弃原有传统系统研究而转向新兴与大模型结合的系统研究。为了适应这一潮流，对于对计算机系统没有兴趣的同学，助教们经过讨论认为需要将重点放在“如何使用好操作系统”和“如何理解操作系统的能力”两个部分。为此，助教们结合 2019、2020、2021、2022 级日常作业的内容，将小作业实践总结的目标进行整合，整理出了这一版实践的目标手册。

由于时间仓促，这一版本内容相对比较单薄。期待同学们在作业发展的过程中不断迭代增加内容。

第一版实践手册牵头讨论和实践的同学是郑文鑫（2017）、王鲲鹏（2022）、房诗涵（2022）、徐子绎（2022）、潘屹（2022）。

第 1 章 总述

本系列作业旨在引导大家通过修改 Linux 内核的实现感受操作系统的功能。作业具体包括以下 6 个大类：启动、系统调用与特权态执行、内存管理、文件系统、网络与外部设备、安全。每个大类进一步根据难度和内容分为基础、实践、设计、综合。前三部分的内容如表1.1所示。综合部分的选题每一个选题横跨两个子项目，内容和目标如表1.2所示。

表 1.1: 分项内容

分项	基础	实践	设计
系统启动	Read ACPI Table	Hack ACPI Table	UEFI 运行时服务
系统调用 特权态	用户定义的系统调用	vDSO	无需中断的系统调用
内存管理	页表和文件页	内存文件系统	内存压力导向的内存管理
文件系统	inode 和扩展属性管理	FUSE	用户空间下的内存磁盘
网络 外部设备	tcpdump 和 socket 管理	NCCL	DPDK

表 1.2: 综合部分分项内容

项目	涉及分项
使用 RDMA 的远程内存 Swap	内存管理、网络与外部设备
用户程序嵌入内核执行	系统调用、内存管理
内核空间下的内存磁盘	内存管理、文件系统
机密虚拟机 Trapless 的宿主协同内存管理	内存管理、安全与虚拟化

第 2 章 项目环境准备

2.1 基准代码

本系列的基准代码规则如下：

1. **Linux**: 任何体系结构下高于 5.10.20 版本的 Linux 内核都是可行的选项，你可以从https://www.kernel.org/中选择合适的版本。对于 `initramfs`，你可以选择任何一个发行版。如果你选择直接使用 qcow 或者其他完全预制好的 Ubuntu 镜像，你需要手动在内部更换内核以确保你的代码能够生效。
2. **QEMU**: 版本 >=7。
3. **EDK2**: 跟随https://github.com/tianocore/edk2选择合适的版本。其中 Ubuntu_GCC5 并不意味着 GCC 版本必须是 5。

2.2 在 QEMU 中启动 Linux 镜像

注 简单起见你可以直接使用预制作好的 `qcow` Ubuntu 镜像作为磁盘镜像由 qemu 直接引导，具体教程：https://documentation.ubuntu.com/public-images/en/latest/public-images-how-to/launch-qcow-with-qemu/

TODO: https://blog.csdn.net/benkaoya/article/details/129469116

第 3 章 系统启动

计算机上电后，系统会进行一系列初始化操作完成硬件设备的逐个启动。早年这一个过程由执行每个设备上的 ROM 中存储的代码（一般称为固件 firmware）结合 BIOS 完成。随后，更加通用的 UEFI 被提出，UEFI 的全称是 Unified Extensible Firmware Interface，它是一套统一使用 C 语言编写在 OS 启动前的固件代码段。

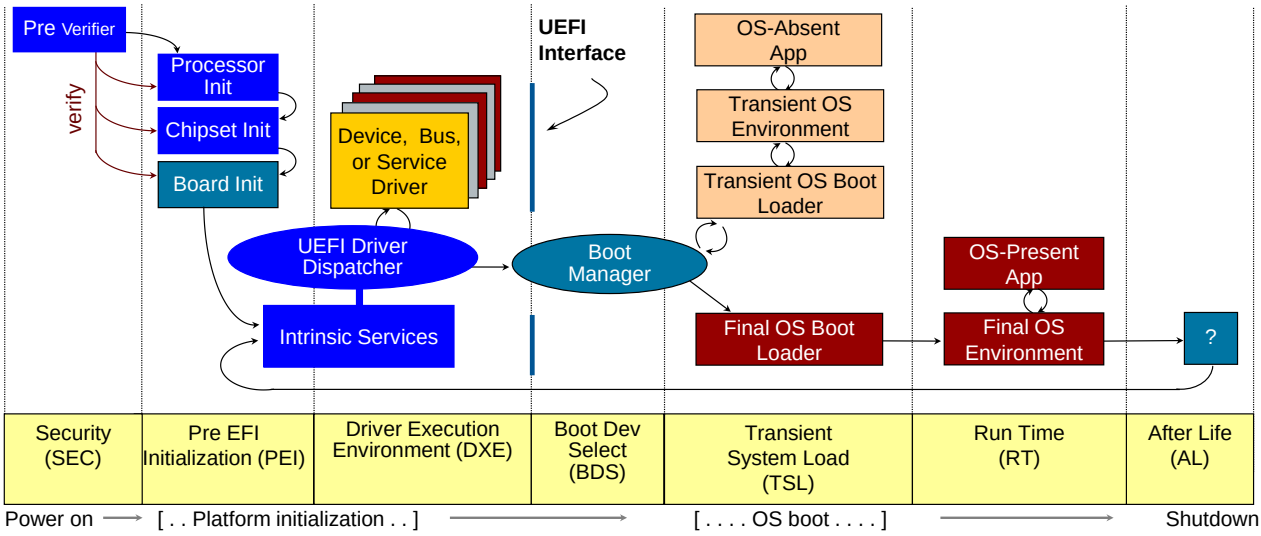

图 3.1: UEFI 启动过程

这些固件代码段还会收集硬件信息，并且移交给操作系统。早年的黑苹果、伪造 OEM 激活 Windows 7 都采用劫持 UEFI 启动后 ACPI Table 内容实现欺骗 OS 的效果。在这个专题的实践中，我们需要体验读取、修改系统启动过程中的各个表，并且尝试传递更多的信息给操作系统。

3.1 Read ACPI Table

目标：修改 EDK2 的 UEFI 组件，在其中增加函数：`PrintAllACPITables`。该函数不传入任何参数，要求打印每一张表的地址、长度以及每一张其中所有的信息与校验信息。

可以参考 EDK2 下的 `AcpiView`，但是不得直接复制。

3.2 Hack ACPI Table

目标：修改 EDK2 的 UEFI 组件，在其中增加函数：`ChangeACPITable`。该函数传入一个 `UINTN` 的对象表明修改第几张表、输入 `EFI_ACPI_DESCRIPTION_HEADER*` 表明待修改的数值，要求修改完成后打印每一张表的地址、长度以及每一张其中所有的信息与校验信息。

测试：通过引导 Linux 后读取对应的 ACPI 表检查输出进行验证。

3.3 UEFI 运行时服务

目标：增加一个新的 UEFI 运行时服务，使得 Linux 系统可以调用这个新的服务导出硬件运行时信息。

提示：需要修改 UEFI 固件和 Linux 内核（sysfs 部分），方便直接读取导出的结果。

测试：展示

第 4 章 特权态与系统调用

操作系统对特权态的管理是操作系统的重要组成部分，影响了用户态程序的编写。这些系统调用也是操作系统向用户态提供抽象的重要部分，例如 Linux 中将设备的读写都转换为文件读写，因此对应的系统调用都和文件的操作读写相关。Windows 的系统调用接口则更为复杂一些，因为其内核的构成并不是单纯的宏内核，对于最小执行单元也有不同的定义。在这个专题的实践中，我们的目标是体验操作系统内核在通过系统调用进行系统管理上的重要作用。

4.1 用户定义的系统调用

目标：新增内核的键值存储模块（Key-Value Store），要求提供 read 接口和 write 接口。内核的键值数据需要存储在进程相关的结构体中，键值存储的对象是进程为对象。

提示：修改用户态系统调用的二进制文件和内核的系统调用接收器。

测试：要求暴露给程序的系统调用接口为 `read_kv(int k);` `write_kv(int k, int v);`。进行多线程并行读取修改。

4.2 vDSO

在上一个体验中，你已经增加了系统调用感受到了系统的服务。然而每次系统调用还是需要通过异常或者中断进行特权态切换，会带来较大切换开销。经过观察，可以发现一部分系统调用并不会泄露内核的状态也不会修改内核。对于这些系统调用可以在用户态以只读共享页面的方式直接进行而无需进入操作系统内核。Linux 中提供了 vDSO 支持这一想法。

目标：在 vDSO 中新增一个读取当前进程对应的 `task_struct` 中的所有信息。

测试：测试读取的数据是否正确，对于指针对象只需要回传指针地址即可。

4.3 无需中断的系统调用

通过 vDSO，我们可以实现无需特权态切换对只读数据的访问。那么对于需要请求内核服务的系统调用，是否也可以避开特权态切换呢？答案是肯定的，FlexSC [3] 给出了对应的解决方案。这一解决方案可以简化为用户态程序和操作系统内核存在共享页面，用户态程序将系统调用需要的参数写进这个共享页面，然后操作系统内核从这个共享页面中读取数据并且将结果写回该页面实现基于共享内存的系统调用。

目标：在 Linux 内核中实现无特权态切换的系统调用服务。

提示：用户态线程比较容易实现请求后等待结果的自旋，需要确保内核处理的线程始终在线。

测试：系统调用密集的程序的吞吐（该系统调用实现应当对用户无感）。

第 5 章 内存管理

在计算机系统（1）中，我们在虚拟存储器部分引出了交换空间的概念。管理内存是硬件和软件协同的结果。在 CPU 启动后，内存的地址空间完整提供给第一个启动的软件，也就是操作系统。随后操作系统在创建每一个进程时分配页表进行页映射，从而提供每个进程完整虚拟地址空间的抽象。在这个专题的实践中，我们的目标是体验操作系统中不同的内存类型和对应管理内存的方式。

5.1 页表和文件页

目标：读取页表，并且手动修改页表页的映射实现共享（通过 `mmap` 调整映射，通过修改内核读取页表）；设置文件页实现读写落在文件的指定区域中。

测试：测试页表映射和修改是否生效到文件中。

5.2 内存文件系统

Linux 中提供了内存文件系统(RAMfs)的支持,其有 MMU 版本和 NoMMU 版本,分别位于 `fs/ramfs/file-mmu.c` 和 `fs/ramfs/file-nommu.c`。RAMfs 无法持久化，在系统崩溃时文件会全部丢失。现在你需要为 RAMfs 提供简单的持久化支持，当某个文件被 flush 到 RAMfs 时，该文件同步刷到一个预先设置的同步目录中以持久化该文件。

目标：增加 flush 下的文件持久化。

测试：测试持久化的文件内容语义和多线程持久化。

5.3 内存压力导向的内存管理

MacOS 给出内存压力的概念，相比于 UNIX 的 swap 大小，它被认为是更好的表达虚拟内存健康程度的标志。

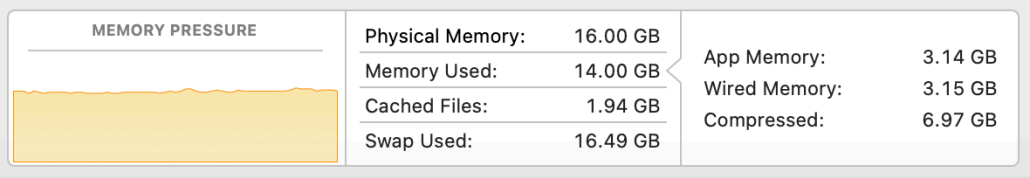

图 5.1: MacOS 中的内存压力

目标：该实践任务包含两个子任务点。

1. 利用 Linux 的 Wired Memory（即 Active Memory）和压缩内存给出一个 Linux 下的内核内存压力指示器。
2. 在用户态划出一块内存空间，实现一个简单的基于内存压力的交换策略。

提示：区分 wired memory 和 inactive memory 的区别。

测试：测试内存压力和感知准确度的对比（演示），测试交换策略。

第 6 章 文件系统

文件是 UNIX 设计哲学中的一个重要组成部分，一切皆文件的抽象使得用户调整系统的参数也变得更加简单。因此，如何实现一个既能管理系统又能管理日常文件的文件系统便显得更加重要。Linux 通过一套统一的 VFS（虚拟文件系统）抽象达成这一目标。在本实践部分，你需要对 VFS 进行修改并且尝试使用文件系统。

6.1 inode 和扩展属性管理

目标：读取设置 inode 的基本信息，引入新的文件扩展属性。

测试：文件扩展属性的读写。

6.2 用户态文件系统

由于文件系统在 Linux 中被编译为内核对象（kernel object），文件系统崩溃会传导到内核造成内核不可用。此外，内核对象的灵活性较差。Linux 给出了 FUSE 的服务，允许在用户态实现一个文件系统。

目标：使用 FUSE 实现 GPT 服务。声明一个 GPTfs，其中的目录为每一个对话 session，对话 session 文件夹下的 input 为本轮用户输入的 prompt，output 文件为 GPT 输出的结果。每一轮对话都是单轮对话，无需考虑记录上下文。

测试：基本的读写效果

6.3 用户空间下的内存磁盘

随着硬件的发展，内存的价格已经显著下降。内存的访问速度非常快，可以作为一部分反复读写的文件的临时存储。在这一个任务中，你需要从用户态划出一块区域用于磁盘存储。

目标：使用 FUSE 实现 RAMfs 功能，要求读写碎片尽可能少并且支持硬链接。

提示：可以参考内核中 RAMfs 的功能，但是不要抄袭。

测试：文件读写和并发读写。

第 7 章 网络与外部设备

7.1 tcpdump 和 socket 管理

Linux 内核中提供了抓包等服务作为内核的网络套件的一部分。

目标：实现如下内容

1. 按照一定条件进行抓包（要求修改 tcpdump，不允许直接用现成的参数）
2. 内核管理 socket 的安全性实现 per-thread 的 socket fairness 管理。

测试：多 Socket 读写的管理（测试 p99 延迟）

7.2 高速通信库：NCCL

现代的高性能计算已经不能由单节点计算满足，需要多个节点多台机器进行并行计算。随着单节点算力的不断上升，并行计算时的通信成为了影响整个计算集群效率的关键因素。传统的网络通信已经不能满足这一通信需求，各个计算卡的厂商普遍提供了自己的高速互联协议（如英伟达的 NCCL、华为的 HCCL）。

目标：通过 NCCL 实现简单的信息传输，比较和传统使用以太网的速度、正确性差异。

7.3 数据平面开发套件：DPDK

当前的网络协议都在内核空间中，因此涉及网络的操作都需要通过特权切换进入内核。对于需要高速通信的程序，反复的上下文切换显然不符合要求。不过，由于完整实现基于 DPDK 的通信所需要的代码量太大，我们在这里进行简化，只需要实现简单的网络即可。

目标：实现用户态基于 DPDK 的网络 QoS 管理，实现多个进程访问网络的带宽平衡。

第 8 章 综合应用

当你来到这一章节时，你应该对 Linux 内核的主要组成部分有了清晰的了解，对每个部分的修改效果应该也熟悉了。在这个基础上，我们进一步探讨我们的综合应用。

8.1 使用 RDMA 的远程内存 Swap

在这个项目中，你需要实现一个简单的使用远程内存的交换分区（swap）。具体而言，当缺页异常发生时，你需要使用 RDMA 访问远程内存将页面换到本地。如果远程页面不存在，你需要寻找本地交换文件中寻找这个目标页面。在两者都不存在的情况下，应当给出一个内存错误（如同访问非法地址的错误）。

8.1.1 指标

项目形态。 你的项目最终的形态应当符合以下指标：

1. 将交换分区的内存页面通过 RDMA 存储到远程内存上。
2. 设定一个可调整的比例决定远程内存和本地内存的比例。
3. 对应用无感。

基础评测指标。 项目的基础评测指标如下：

1. 使用 `memcached` [2] 存储一系列数据（YCSB [1]）在使用 50% 远程内存时在 50% 能够产生一个显著的时间差。
2. 使用 RDMA 的交换时间需要比磁盘交换更短。

对于基础评测指标，一个简单的示意图如图 8.1所示。

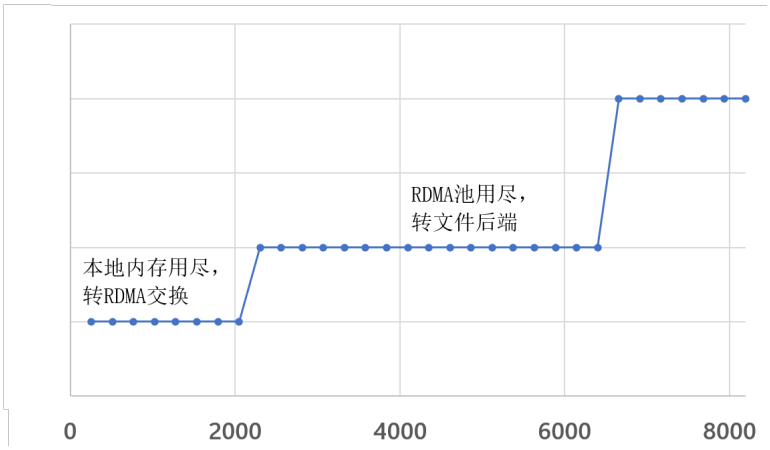

图 8.1: RDMA 和文件混合交换分区下的内存访问量同时间的示意图。图中假设本地内存为 2048MB，RDMA 内存约 4096MB。相对时间仅为示意图，不代表真实场景数据。

进阶指标。 在基础评测的指标上，结合内存访问的特点，实现页预取和交换，目标发生尽可能少的缺页异常。

8.1.2 快速上手和实现重点提示

本项目的核心包括两个部分，内存管理和 RDMA 的访问。对于内存管理部分，需要修改原有系统的 Page Fault 处理流程。RDMA 部分则需要维护 RDMA 连接确保发生缺页异常时向远端设备发起 RDMA 读写请求。

内存管理。 缺页异常发生后，操作系统需要确认这个页位于什么地方（是在内存中但是页表没有映射？还是这个页已经被交换到磁盘设备？或是这个页根本不存在？）。传统的顺序流程可以在体系结构下的 `mm/fault.c` 中找到相关的代码。加入 RDMA 后，需要保证 RDMA 部分不能被换出，并且加入对 RDMA 内存的搜索。

RDMA 管理。 RDMA 的通信与不同的以太网通信不同。为了保证传输的速度，RDMA 没有使用套接字（Socket）抽象。取而代之的是两大类 Verbs（Memory Verbs 和 Message Verbs）这两大类的 Verbs 具体包含：Read（远程主机读取部分内存）、Write（数据写到远端主机）、Atomic（原子操作）；Send（把数据发送到远程 QP 的接收队列里）、Receive（接收主机被告知接收到数据缓冲）。具体的管理可以参考以下链接：（中文）https://zhuanlan.zhihu.com/p/649468433（英语官方原版）https://academy.nvidia.com/en/course/rdma-programming-intro/?cm=446

特殊点提示。

- 多线程的访问安全性：多个进程同时发生缺页异常在远端的处理过程；
- RDMA 内存的管理：减少远端内存中的内存碎片。

8.2 用户程序嵌入内核执行

用户程序执行时需要通过系统调用（syscall）调用特权系统服务，每一次系统调用都有上下文切换的开销，从而使得密集调用系统调用的程序产生额外的开销。一种简单的优化方法是将含有系统调用的程序嵌入内核执行，从而避免上下文切换。这一更改实际上给内核打了很多的“洞”，忽略了内核的权限隔离，因此是不安全的。我们深入之后会发现，这一想法的核心其实是一部分上下文权限隔离对程序而言过强了，程序没有用到对应的功能因此也不需要对应的隔离。在这个基础上，我们可以通过为一些简单程序设计内核中隔离他们程序的方案。具体而言，通过修改 fork 的过程将程序直接嵌入内核空间，增加一些特殊的机制（你自定义的机制，可以是页表、MPK 等）实现该段代码在内核中仅能访问特定内存（相当于自己的地址空间）。假设程序不是恶意的，因此不需要你考虑控制流完整性，但是需要注意代码的映射区域。同时假设在这项作业中程序已经被编译为位置无关代码。

8.2.1 指标

项目形态。 你的项目最终的形态应当符合以下指标：

1. 将一段程序嵌入到内核态执行，程序本身无需修改。
2. 程序仅能访问自己的内存区间，而不能访问内核的其他数据结构。

基础评测指标。 项目的基础评测指标如下：

1. 一段仅包含一系列系统调用的程序可以进入内核实现吞吐的显著提升。
2. 程序访问非法内存应当退出当前程序的执行，给出错误信息（可以在内核日志）并且不影响当前内核的执行。
3. 支持多线程程序的执行，不影响多线程程序的可扩展性。

进阶指标。 在基础评测的指标上，支持带有多个用户库文件的用户程序进入内核执行。

8.2.2 快速上手和实现重点提示

本项目的核心包括两个部分，代码启动阶段（fork）和执行阶段（exec）。

启动阶段 启动阶段用户程序并没有执行，核心需要考虑的部分是如何映射用户段的代码到内核空间，以及如何设置权限隔离的初始状态。此外，需要在启动阶段拦截系统调用，确保系统调用转换为内核态的函数跳转。

执行阶段 执行阶段最核心的是在遇到错误的时候如何修正错误的状态，从而避免嵌入内核的程序崩溃内核。

特殊点提示。

- 多线程执行：与内核线程的绑定执行关系；
- 内核状态的保护：防止单个程序崩溃整个内核。

8.3 内核空间下的内存磁盘

随着硬件的发展，内存的价格已经显著下降。内存的访问速度非常快，可以作为一部分反复读写的文件的临时存储。这一部分是用户空间下的内存磁盘的进一步延伸。

8.3.1 指标

项目形态。 你的项目最终的形态应当符合以下指标：

1. 将一段内存空间作为磁盘空间映射给内核。
2. 映射的磁盘空间可以进行文件读写和文件夹创建。
3. 注册为一个磁盘设备而非文件系统。

基础评测指标。 项目的基础评测指标如下：

1. 使用 `dd` 一次性创建一个大文件的开销足够小（需要接近内存的读写速度）。
2. 多次创建删除文件之后实际存储容量始终保持一致。

进阶指标。 在基础评测的指标上，减小文件元数据的开销大小。

8.3.2 快速上手和实现重点提示

本项目的核心是内存空间的映射和分区管理。与映射在用户空间不同，内核空间的映射需要保证用于虚拟磁盘的数据不能破坏内核数据的正确性。此外，用户空间可以通过 `fuse` 实现用户态的分区访问。在内核中，该模块需要实现为内核模块，因此需要符合内核的一系列数据接口。

特殊点提示。

- 多线程执行：多线程读写保证数据的一致性。
- 文件碎片：文件的碎片应该尽可能少。

8.4 机密虚拟机 Trapless 的宿主协同内存管理

机密虚拟机的一个核心是内存需要加密以防止宿主可以获取到虚拟机内存造成数据泄露，然而这样的安全措施也造成了宿主机无法感知虚拟机内的内存情况，内存分配无法最优化。一种方式是虚拟机内核主动向宿主机的虚拟机管理器发起调用引导宿主机按照一定的偏好管理内存。这种方式需要下陷到宿主机的虚拟机管理器，有较大的切换开销。请设计一种机制，以不下陷的方式引导宿主机的按照一定偏好管理内存。

8.4.1 指标

项目形态。 你的项目最终的形态应当符合以下指标：

1. 不修改虚拟机内应用程序，允许修改虚拟机管理器和虚拟机内的内核。
2. 指示内存偏好时不发生下陷。

基础评测指标。 项目的基础评测指标如下：

1. 使用内存密集型任务在机密虚拟机内部设置特定内存位置后计算速度和用户态的读写速度一致。

8.4.2 快速上手和实现重点提示

本项目的指标较少，因为核心代码需要较大的修改，包括三个部分：宿主机的内核、虚拟机的内核、宿主机的虚拟机管理器。宿主机内核需要设定特殊的接口管理加密的内存页面。虚拟机内核需要修改实现无下陷与宿主机的虚拟机管理器进行交互。宿主机的虚拟机内存管理器需要增加接收虚拟机内核的特殊内存偏好参数并且传递给宿主机。

特殊点提示。

- 虚拟机下陷的处理。
- 共享内存页的管理。
- 宿主机的内存管理。

参考文献

[1] Brian F Cooper et al. “Benchmarking cloud serving systems with YCSB”. In: *Proceedings of the 1st ACM symposium on Cloud computing*. 2010, pp. 143–154.

[2] Brad Fitzpatrick. “Distributed caching with memcached”. In: *Linux journal* 2004.124 (2004), p. 5.

[3] Livio Soares and Michael Stumm. “{FlexSC}: Flexible system call scheduling with {Exception-Less} system calls”. In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. 2010.