虚拟化课程实验 2023秋

概览

本次实验为单人作业，实验内容包括以下四个部分：

1. CPU 虚拟化实验：使用 KVM 接口运行最简化 QEMU；
2. 内存虚拟化实验：打印虚拟机的内存地址翻译（GVA->HPA）过程；
3. I/O 虚拟化：访问虚拟设备，实现自定义的功能；
4. StratoVirt 与 QEMU 方案的对比；

实验环境

实验需要 Linux 支持 KVM，即存在 `/dev/kvm` 文件，因此需要在物理机环境下进行；

本实验主要在 **x86** 平台进行。

重要链接

- 课程的参考书籍为 《深入浅出系统虚拟化：原理与实践》；
- 实验的代码：https://github.com/GiantVM/Book，请 clone 该仓库；
- StratoVirt 平台：https://gitee.com/openeuler/stratovirt，是最后一部分的轻量级虚拟机对比方案；
- QEMU 源码：https://github.com/qemu/qemu，实验三涉及到需要修改并重新编译 QEMU 的内容，目前助教在 6.2 和 7.2 版本上均可以完成实验；
- 答疑汇总：虚拟化课程实验 Q&A

CPU 虚拟化

该部分对应 《深入浅出系统虚拟化：原理与实践》 的 2.3.5 节的实验，请先学习该部分的内容，实验代码见：https://github.com/GiantVM/Book/tree/master/Chapter-2

调研环节

请在报告中回答以下问题：

1. 请调研并用你的理解解释可虚拟化架构与不可虚拟化架构的概念（参考书籍 2.1 节）
2. 请基于你的理解谈谈虚拟化的“陷入再模拟”的概念（参考书籍 1.3.3 节）
3. 请调研并用你的理解解释 Intel VT-x 的特权级是如何划分的。这种非根模式为何有助于 Hypervisor “陷入再模拟”（参考书籍2.2节）？

实验环节

实验说明

- 本实验对 Linux 内核版本没有要求，只需要支持 KVM 的机器即可；
- 下载仓库，进入 "Chapter-2" 目录下，文件 `sample-qemu.c` 实现了一个类似于 QEMU 的小程序，在 **x86 平台**通过调用 **KVM** 接口，执行指定二进制代码输出 “Hello, World!”；
- 可以通过 `$ make` 指令编译目标程序，运行后可以在终端中看到输出结果；
- 该实验在通过 ioctl 调用 KVM 创建 VM、设置虚拟 CPU、设置虚拟内存、设置虚拟机寄存器内容后，在 while 循环中调用 KVM_RUN 指令进入 VM 中开始执行预设的写在 `code` 数组中的二进制代码，并在触发部分特权指令（如 `code` 变量中使用的 out 汇编指令）后，KVM 退出，进入到“陷入再模拟”阶段，可以根据 KVM 的退出原因以及相应参数进行自定义的模拟，直到处理完成后进入下一个 KVM_RUN 的阶段使客户机继续运行。

实验目标

- 理解了上述流程后，请着重关注 KVM_EXIT_IO 的模拟过程，**并根据你的理解重写该部分的 “陷入再模拟” 过程**，使得运行时输出到终端的 “Hello World!” 中的小写字符变为输出大写字符 “HELLO WORLD!”；
- **请将修改后的代码版本以 "modified-qemu.c" 文件名与报告一并提交**；

内存虚拟化

该部分对应 《深入浅出系统虚拟化：原理与实践》 的 3.3.2 和 3.3.4 节的实验，请先学习该部分的内容，实验代码见：https://github.com/GiantVM/Book/tree/master/Chapter-3

调研环节

请在报告中回答以下问题：

1. 虚拟内存到物理内存的映射是如何通过页表实现的？（参考教材 3.2.1 节）
2. 请阐述**影子页表**（SPT）和**扩展页表**（EPT）各自的优缺点（参考教材 3.2.2-3.2.3 节）
3. （**选做，不扣分**）请根据你的理解解释 QEMU是如何通过 KVM 接口建立 EPT 的（参考教材第三章图 3-12 ）

实验环节

实验说明

本次实验的内存虚拟化部分基于 Linux 4.18/4.19 版本内核

1. 如果自行准备环境则需要在实验的主机上更换内核，实验的客户机可以使用 ubuntu 提供的 cloud-image 来完成，客户机使用恰当版本时无须更换内核。
2. 如果在服务器上进行实验，则需要启动嵌套虚拟化再更换 L1 虚拟机内核。

在 Linux 内核的基础上，实验代码中的三个文件主要增添了一个 Hypercall，具体内容位于 `x86.c` 中 `kvm_emulate_hypercall` 函数中的 case 22，即客户机中可以通过 Hypercall 22 来访问该功能。随机 KVM 会在主机中打印出 EPT 地址翻译过程（GPA-> HPA, 在主机中通过 dmesg 查看）。实验代码中的 `gpt-dump.c` 则是作为内核模块运行在客户机上，也会打印出客户机中内核可见的 GVA -> GPA 地址翻译过程（在 gpt-dump.txt 中查看）

实验目标

这一部分的任务是：

1. 下载 Linux 4.18/4.19 版本内核，将 `arch/x86/kvm/` 目录下的 `x86.c` ， `mmu.h` ， `mmu.c` 三个文件替换为仓库第三章目录下的相应文件，编译并替换主机内核；
2. 构建客户机环境，更换为 4.18/4.19 版本内核；
3. 在客户机中下载实验代码，进入 “Chapter-3” 目录下，编译内核模块并运行，观察运行结果；
4. 在主机中借助 QEMU 监控器打印客户机 MemoryRegion 树结构；

请将各个部分完成后的情况截图并写在报告里，完成实验后请将 `gpt-dump.txt` 的内容以及主机执行 `$ dmesg` 命令后打印的 EPT 翻译的内容记录到报告里，并分析从内存地址从 GVA 到 HPA 的地址翻译过程

更换主机内核

该步骤如果在服务器做的同学，需要使用嵌套虚拟化环境进行实验，需要确保主机的 kvm_intel 模块已使用 nested=1 参数，并在启动第一层虚拟化实验环境时为 QEMU 添加 `-cpu host` 选项

```
# 下载 linux 内核
git clone https://github.com/torvalds/linux.git
# 4.18 版本也可
git checkout v4.19

# 替换 Linux arch/x86/kvm 目录下的三个文件为实验代码中的 kvm 目录下，过程略

cp /boot/config-<some version> .config
# 有个 pem 的报错，可以在 .config 中搜索两处 "debian/" 并替换为空字符串
make menuconfig
# 确保存在： Virtualization Support -> KVM for Intel, KVM for AMD

# 使用 gcc-8，避免版本问题，高版本可能无法通过 apt 直接安装，可选的通过添加 focal 的 apt 源来获取
# error: ‘-mindirect-branch’ and ‘-fcf-protection’ are not compatible
sudo apt install gcc-8
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 90
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 100
sudo update-alternatives --config gcc

# 编译内核
make
# INSTALL_MOD_STRIP=1 用于缩小 initrd 大小
sudo make INSTALL_MOD_STRIP=1 modules_install
sudo make install
reboot
# 重启如果出现了 has invalid signature 问题则在需要 BIOS 里禁用 secure boot
# 如果出现 junk in compressed archive 问题，这是因为低版本内核不支持 zstd 解压，需要修改/etc/initramfs-tools/initramfs.conf 并 update

uanme -a
# 此时应该可以看到主机的 Linux 版本为相应版本，应在此处截图
```

建立客户机

请在更换好内核版本的主机中完成如下操作：

```
# Guest OS Linux v4.19
# Ubuntu Cloud image: https://cloud-images.ubuntu.com/
wget https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img
qemu-img info bionic-server-cloudimg-amd64.img
qemu-img resize bionic-server-cloudimg-amd64.img 10G

# Cloud Image Config
vi cloud-config
```

```
9
10 # content start，开始复制以下内容直到 # content end，#cloud-config 也要复制
11 #cloud-config
12 password: Virtlabs
13 chpasswd: { expire: False }
14 ssh_pwauth: True
15
16 # content end
17
18 sudo cloud-localds ./seed.iso cloud-config
```

运行客户机

请在主机中运行 QEMU 命令行启动客户机，检查客户机内核版本是否符合要求：

```
1  # 启动 qemu，如果无法通过该 image 登录，可尝试其他版本或搜索 cloud image 用法
2  sudo qemu-system-x86_64 --enable-kvm -m 2G -net nic \
3      -net user,hostfwd=tcp::2222-:22 \
4      -drive file=./bionic-server-cloudimg-amd64.img,if=virtio \
5      -drive file=seed.iso,if=virtio \
6      -nographic
7
8  # 客户机：  username: ubuntu  password: Virtlabs
9  # 检查客户机当前内核版本
10 uname -a
11
12 # 改变内核版本
13 sudo apt update
14 sudo apt install -y linux-headers-4.18.0-25-generic linux-image-4.18.0-25-
   generic make git gcc
15
16 # 重启客户机再次检查内核版本
17 sudo reboot
18 uname -a
19 # 此时应该可以看到内核版本变为 4.18，  应在此处截图
```

查看地址翻译过程

请在客户机和主机中完成实验，验证 EPT 翻译的过程：

```
1 # 在客户机进行实验
2 git clone https://github.com/GiantVM/Book.git
3 cd Book/Chapter-3
4 make
```

```
# gpt-dump.ko should be here

sudo ./run.sh
# 应在此处保存 gpt-dump.txt 中的内容，为客户机页表的翻译过程
# 若提示 Command not found，请执行 sudo chmod +x ./run.sh 为脚本添加可执行权限后再运行

# 退出客户机：
sudo poweroff

# 在主机运行以下命令，应该可以看到 EPT 翻译过程
sudo dmesg
# 应在此处保存主机中 EPT 的翻译过程的内容
```

打印 QEMU MemoryRegion 树

请在主机中运行 QEMU 命令行启动客户机，并通过 QEMU 监控器打印客户机的 MemoryRegion 树

```
# 在主机中完成实验
# 启动 qemu，注意与之前代码有不同之处
# -monitor stdio 参数将启动 qemu monitor 并重定向到字符设备 stdio，即当前命令行
sudo qemu-system-x86_64 --enable-kvm -m 2G -net nic \
    -net user,hostfwd=tcp::2222-:22 \
    -drive file=./bionic-server-cloudimg-amd64.img,if=virtio \
    -drive file=seed.iso,if=virtio \
    -monitor stdio

# 此时命令行将进入 qemu monitor，可以通过 help 查看支持的命令
(qemu) help

# 使用命令 info mtree 来打印此客户机的 MemoryRegion 树，不同宽度的缩进表示不同树的深度，应在此处截图
(qemu) info mtree
```

进行实验分析

- 请根据你记录的翻译过程进行分析，描述 GVA 到 HPA 的翻译过程，并写在报告中；
- 请根据你的客户机 MemoryRegion 树结构，参考教材 3.3.1 节的内容和图 3-10，画出客户机的 MR 树根 `memory-region: system` 的结构；

I/O 虚拟化

该部分对应《深入浅出系统虚拟化：原理与实践》的 4.3.4 节的实验，请先学习该部分的内容，实验代码见：https://github.com/GiantVM/Book/tree/master/Chapter-4

调研环节

请在报告中回答以下问题：

1. 请阐述 x86 中 MMIO 与 PIO 两种 I/O 方式的特点（参考教材 4.2.2 节）；
2. 请简要描述 virtio 设备在进行 I/O 操作时的工作原理，这样的半虚拟化架构有什么优点？（参考教材 4.2.3 节）
3. （**选做，不扣分**）请调研设备枚举过程并根据你的理解回答设备是如何被发现的（参考书籍的 4.2 节，以及 UEFI 相关内容）；

实验环节

本次实验对 Linux 版本没有特别要求，可以复用之前的客户机环境。

实验目标

- 在 QEMU 中添加模拟的 edu 设备，在客户机环境安装对应驱动并使用测试程序访问虚拟设备，请将客户机中的 log 保存到报告中；
- 基于 QEMU 模拟 edu 设备的实现原理，为 edu 设备添加一项功能；

启动客户机

请在主机上创建 QEMU 客户机，并添加 edu 设备，可以复用之前实验的客户机镜像文件，注意 QEMU 启动的脚本有所区别：

```
# 在主机上执行:
# 可选的 cloud image 搭建环节
wget https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img
qemu-img info bionic-server-cloudimg-amd64.img
qemu-img resize bionic-server-cloudimg-amd64.img 10G

# cloud-config 可以复用内存虚拟化部分的同名文件
sudo cloud-localds ./seed.iso clould-config

sudo qemu-system-x86_64  -smp 2 -m 2G -enable-kvm \
    -drive file=./bionic-server-cloudimg-amd64.img,if=virtio \
    -drive file=seed.iso,if=virtio \
    -device edu,dma_mask=0xFFFFFFFF \
```

```
    -net nic \
    -net user,hostfwd=tcp::2222-:22 \
    -nographic
```

安装 edu 设备驱动

请在客户机内编译并安装 edu 设备驱动：

```
# 在客户机内执行：
# 配置 Guest 环境
uname -a
sudo apt update
sudo apt install -y git gcc make tmux libelf-dev

# 克隆仓库，切换到实验代码 Chapter-4 目录下进行编译
git clone https://github.com/GiantVM/Book.git
cd /Book/Chapter-4
make

# 安装设备驱动并查看设备状态
sudo insmod pci.ko

# 查看 PCI 设备
lspci
# 结果中应该会看到一行：
# 00:04.0 Unclassified device [00ff]: Device 1234:11e8 (rev 10)
# 同样的 QEMU 命令启动，应该是同样的设备地址，运行如下命令查看详细信息
lspci -s 00:04.0 -vvv -xxxx
# 应该可以看到 kernel drvier in use 一项

# 此时安装了驱动，但是还没有在 Linux 中注册设备节点，即看不到 /dev/edu 文件
# 查看驱动信息，以及主设备号，其中的数字为主设备号
cat /proc/devices | grep edu
# 如：  245 pci-edu

# 构造设备节点，<major> 为你看到的主设备号
sudo mknod /dev/edu c <major> 0

# 此后应该可以看到 /dev/edu 文件了
ls /dev/edu

# 清空 dmesg，运行测试代码
sudo dmesg --clear
sudo ./test

# 请将输出内容保存到报告中
```

```
39  sudo dmesg
```

修改 edu 设备

QEMU 中默认的 edu 设备实现了很简单的功能，并支持通过 ioctl 来调用这些功能，其具体的内容可以参考教材 4.3.4 小节的内容。

- 代码 `Chapter-4/test.c` 中体现了 edu 驱动的调用方法，通过指定不同的 ioctl 参数，以调用不同的 edu 驱动功能；
- `Chapter-4/pci.c` 的驱动代码中，则实现了 edu 驱动的 write、read 和 ioctl 等功能，其中 `edu_ioctl` 函数根据传入参数的不同，将会对 edu 设备执行不同的 MMIO 操作；
- 这些 MMIO 操作最终都会在 QEMU 的模拟 edu 设备中进行处理，其实现可以在 QEMU 源码的 `hw/misc/edu.c` 文件中找到，其中最主要的是 `edu_mmio_write` 和 `edu_mmio_read` 两个函数，根据 MMIO 请求的地址不同，将完成不同的设备操作；

请参考上述过程，修改 `Chapter-4/test.c` 、 `Chapter-4/pci.c` 以及QEMU 源代码的 `edu_mmio_write` 或 `edu_mmio_read` 函数（需要重新编译 QEMU），为 edu 模拟设备额外添加一个简单的功能（如打印指定信息，简单的数值运算等）。

请将修改后的代码命名为 `*_modified.c` 和报告一同提交，并在报告中简单描述实现的过程，附上可以体现修改后运行结果的截图。

StratoVirt 与 QEMU 方案对比

StratoVirt 作为一个使用 rust 语言编写的轻量级虚拟化平台，相对 QEMU 删减了许多不常用的功能与设备模型，并针对轻量化这一特性进行优化。该部分的内容为对比 StratoVirt 与 QEMU 两个虚拟化解决方案的各项性能指标。

调研环节

请在报告中回答以下问题：

1. Linux 启动的协议以及 E820 表是如何设计的？（参考教材 6.4.5 节）
2. 请简述 Epoll 实现原理？（参考教材 6.4.7 节）
3. 请执行以下脚本并简述 StratoVirt 的技术重点
   a. 本部分依赖于 shell 环境，选取一个空文件夹即可，脚本为：QA.sh
   b. 该脚本基于 StratoVirt 的 gitee 仓库中的 mini_stratovirt_edu 分支进行，并基于不同 commit 由简到繁构建系统，可以从代码角度以问答形式帮你了解 StratoVirt 的技术重点

```
```

```
chmod +x QA.sh

./QA.sh
```

实验环节

实验目标

你需要对比两种虚拟化平台的以下方面：

- Linux 启动时间的对比
- StratoVirt 与 QEMU 内存占用对比
- I/O 速度对比

请将实验数据记录在报告中，并分析两种平台的轻量化程度

StratoVirti 环境搭建

StratoVirt 的构建可以参考其代码仓库和官方文档：https://gitee.com/openeuler/stratovirt

主要包括以下内容：

- Rust 和 Cargo 环境的准备
- StratoVirt 源码下载与编译

启动时间对比

请在 StratoVirt 客户机和 QEMU 客户机中分别记录启动时间，要求两种方案各启动 10 次记录结果。

StratoVirt：

```
# StratoVirt 启动 OpenEuler
# 获取内核与 RootFS
# 若有网络问题可以使用国内镜像 https://repo.huaweicloud.com/openeuler/openEuler-21.03/stratovirt_img/x86_64/openEuler-21.03-stratovirt-x86_64.img.xz
# 或浏览器访问 https://archives.openeuler.openatom.cn/openEuler-21.03/stratovirt_img/x86_64/ 下载对应文件
wget https://archives.openeuler.openatom.cn/openEuler-21.03/stratovirt_img/x86_64/openEuler-21.03-stratovirt-x86_64.img.xz
wget https://archives.openeuler.openatom.cn/openEuler-21.03/stratovirt_img/x86_64/vmlinux.bin

# 解压镜像文件
```

```
xz -d openEuler-21.03-stratovirt-x86_64.img.xz

# 使用 microvm 启动，注意 <path/to/stratovirt> 应当替换为你编译后的 stratovirt 源代码目录下的 target/release/stratovirt 可执行文件，或者将 stratovirt 添加到环境变量中
<path/to/stratovirt> \
    -machine microvm \
    -smp 1 \
    -m 1024 \
    -kernel ./vmlinux.bin \
    -append "console=ttyS0 root=/dev/vda reboot=k panic=1" \
    -drive file=./openEuler-21.03-stratovirt-x86_64.img,id=rootfs,readonly=off \
    -device virtio-blk-device,drive=rootfs,id=rootfs \
    -qmp unix:stratovirt.sock,server,nowait \
    -serial stdio
# username: root  password: openEuler12#$
# reboot to quit

# 在客户机中查看内核的启动时间并记录在报告中
systemd-analyze time
```

QEMU：

```
# QEMU 启动 OpenEuler
wget https://archives.openeuler.openatom.cn/openEuler-21.03/virtual_machine_img/x86_64/openEuler-21.03-x86_64.qcow2.xz

# 解压镜像文件
unxz openEuler-21.03-x86_64.qcow2.xz

# 使用 qemu 启动
qemu-system-x86_64 \
    -machine q35 \
    -smp 1 \
    -m 1024 \
    -drive file=openEuler-21.03-x86_64.qcow2,if=virtio \
    -qmp unix:stratovirt.sock,server,nowait \
    --enable-kvm \
    -serial stdio
# 如果提示没有 SDL 无法启动可以将 -serial stdio 替换为 -curses 选项
# username: root  password: openEuler12#$
# 使用 poweroff 命令来退出
# 使用 reboot 命令来重启

# 查看启动时间并记录在报告中
```

```
systemd-analyze time
```

内存占用对比

请在主机中记录同样使用 1G 内存时，使用 pmap 记录 StratoVirt 与 QEMU 进程各自占用的内存数量，要求两种方案各启动 10 次记录结果。

StratoVirt：

```
# 启动 StratoVirt
<path/to/stratovirt> \
    -machine microvm \
    -smp 1 \
    -m 1024 \
    -kernel ./vmlinux.bin \
    -append "console=ttyS0 root=/dev/vda reboot=k panic=1" \
    -drive file=./openEuler-21.03-stratovirt-x86_64.img,id=rootfs,readonly=off \
    -device virtio-blk-device,drive=rootfs,id=rootfs \
    -qmp unix:stratovirt.sock,server,nowait \
    -serial stdio

# 在主机使用 pmap 查看进程占用的内存
sudo pmap -x $(pgrep stratovirt)
```

QEMU：

```
# 启动 QEMU，-cpu host 是为了 CPU 模型兼容性考虑，否则 fio 会有 core dump 问题
qemu-system-x86_64 \
    -machine q35 \
    -smp 1 \
    -cpu host \
    -m 1024 \
    -drive file=openEuler-21.03-x86_64.qcow2,if=virtio \
    -qmp unix:stratovirt.sock,server,nowait \
    --enable-kvm \
    -serial stdio

 # 使用 pmap 查看进程占用的内存
sudo pmap -x $(pgrep qemu)
```

CPU 性能对比

软件环境配置

本部分实验需要用到性能测试工具 sysbench，若客户机没有网络，需要通过挂载的方式将 rpm 包拷贝到客户机镜像中，之后在客户机内部利用 rpm 安装，需要下载 sysbench_src 文件夹。

`openEuler-21.03-stratovirt-x86_64.img` 中添加 sysbench_src 安装包

```
# 在主机上运行
# 挂载镜像文件
sudo su
mkdir mnt
mount openEuler-21.03-stratovirt-x86_64.img  mnt/

# 将 sysbench_scr 文件夹拷贝到挂载的镜像中，此处 <sysbench_src path> 应为你的文件路径
cd ./mnt/root
cp -r <sysbench_src path> .

# 结束挂载
cd ../../
fuser -m -v mnt
umount ./mnt
rm -rf ./mnt
```

`openEuler-21.03-stratovirt-x86_64.qcow2` 中添加 fio

```
# 在主机上运行
# 挂载镜像文件
sudo su
mkdir mnt
guestmount -a openEuler-21.03-x86_64.qcow2 -m /dev/sda1 mnt

# 将 sysbench_scr 文件夹拷贝到挂载的镜像中，此处 <sysbench_src path> 应为你的文件路径
cd ./mnt/root
cp -r <sysbench_src path> .

# 结束挂载
cd ../../
guestunmount ./mnt

ls mnt
# 应该为空了
rm -rf mnt
```

安装 sysbench：

```
# 启动虚拟机 切换到sysbench_src目录下
cd sysbench_src

# 安装sysbench
rpm -ivh *.rpm --nodeps
# 在qemu启动的虚拟机中可能需要逐一安装

sysbench cpu help
# 出现以下内容说明安装成功:
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

cpu options:
  --cpu-max-prime=N upper limit for primes generator [10000]
```

可以通过 `sysbench --help` 指令或网络资料查看 sysbench 的使用方法

要求在客户机中使用 sysbench cpu 测试完成以下测试：

- 线程数依次设置为1，4，16，32，64，测试不同线程情况下的结果；
- 每种类型的测试至少跑两次；

内存性能对比

该部分依旧采用sysbench进行测试，执行 `sysbench memory help` 指令可以看到相应的帮助信息：

```
# sysbench memory help
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

memory options:
  --memory-block-size=SIZE    size of memory block for test [1K]
  --memory-total-size=SIZE    total size of data to transfer [100G]
  --memory-scope=STRING       memory access scope {global,local} [global]
  --memory-hugetlb[=on|off]   allocate memory from HugeTLB pool [off]
  --memory-oper=STRING        type of memory operations {read, write, none} [wri
  --memory-access-mode=STRING memory access mode {seq,rnd} [seq]
```

要求在客户机中使用 sysbench memory 测试以下测试：

- 线程数为4，内存块大小分别设置为1k，2k，4k，8k，测试不同内存块大小的影响
- 每组测试要测试 **顺序访问** 和 **随机访问** 两种情况，每组测试至少执行两次，每次至少 60 秒钟

I/O 性能对比

该部分依旧采用sysbench进行测试，执行 `sysbench fileio help` 指令可以看到相应的帮助信息：

```
# sysbench fileio help
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

fileio options:
  --file-num=N                  number of files to create [128]
  --file-block-size=N           block size to use in all IO operations [16384]
  --file-total-size=SIZE        total size of files to create [2G]
  --file-test-mode=STRING       test mode {seqwr, seqrewr, seqrd, rndrd, rndwr,
  --file-io-mode=STRING         file operations mode {sync,async,mmap} [sync]
  --file-async-backlog=N        number of asynchronous operatons to queue per th
  --file-extra-flags=[LIST,...] list of additional flags to use to open files {s
  --file-fsync-freq=N           do fsync() after this number of requests (0 - do
  --file-fsync-all[=on|off]     do fsync() after each write operation [off]
  --file-fsync-end[=on|off]     do fsync() at the end of test [on]
  --file-fsync-mode=STRING      which method to use for synchronization {fsync,
  --file-merged-requests=N      merge at most this number of IO requests if poss
  --file-rw-ratio=N             reads/writes ratio for combined test [1.5]
```

要求在客户机中使用 sysbench 完成以下几组测试：

- 线程数为1，bs设置为4k，测试模拟单个队列读写的延迟；
- 线程数为32，bs设置为128k，测试吞吐量，跑满整个磁盘带宽；
- 线程数为32，bs设置为4k，测试 IOPS；
- 每组测试至少要测试**随机读、随机写、顺序读、顺序写**四种情况，每种类型的测试**至少跑两次**，每次至少 60 秒钟，请将**测试方式**写入报告，并将结果记录为表格；

QEMU 命令行最好使用针对物理设备的 IO 而非镜像来完成实验，且 QEMU 侧需要禁用缓存。

QEMU磁盘设置参考 `-drive cache=none,if=virtio,id=vdisk1,format=raw,file=[mnt disk path]`

这里的挂载磁盘路径最好是对磁盘重新分区后的一个裸盘，因为读写测试时可能会影响磁盘上的内容

总结

在获取启动时间、内存占用、CPU性能、内存性能和I/O性能数据后，请将数据记录以表格的形式在报告中，并分析 StratoVirt 相比 QEMU 的轻量化方面效果如何，并结合设计思想、编程语言、实现方案等分析为何会有这样的效果。

作业提交

请将报告与源码一并提交到 canvas，命名为 "姓名-学号.zip"，各部分实验截止日期如下：

1. CPU 虚拟化实验（提交截止日期：11月30日 23:59）
2. 内存虚拟化实验（提交截止日期：12月10日 23:59)
3. I/O 虚拟化（提交截止日期：12月24日 23:59)
4. StratoVirt 与 QEMU 方案的对比（提交截止日期：12月31日 23:59)