

# 上海交通大学

SHANGHAI JIAOTONG UNIVERSITY



## 工程实践与科技创新 3-D 作业 2

### QEMU 的编译安装及性能测试

姓名：薛春宇

学号：518021910698

完成时间：2020/9/23

# QEMU 的编译安装及性能测试

薛春宇 518021910698

## 1. 实验目的

- (1) 掌握 QEMU 源码的编译方法，在此基础上安装 QEMU 模拟处理器；
- (2) 在 QEMU 模拟处理器上安装虚拟机；
- (3) 使用 Benchmarks 测试 QEMU 上虚拟机的各项性能，并通过性能比较说明 KVM 模块对虚拟机性能的重要性；
- (4) 适当修改 QEMU/KVM 的源码并运行；

## 2. 实验内容

### 2.1 QEMU 基本原理

QEMU 是一套以 GPL 许可证分发源码的模拟处理器，在 GNU/Linux 平台上使用广泛。通过 qemu-kvm 加速模块，QEMU 能模拟至接近真实电脑的速度。

QEMU 有两种主要运作模式：

- (1) 用户模式：QEMU 能启动那些为不同中央处理器编译的 Linux 程序；
- (2) 系统模式：QEMU 能模拟整个电脑，包括中央处理器及其他周边设备，使得为跨平台编写的程序进行测试及除错工作变容易，也能用来在一部主机上虚拟多部不同的虚拟电脑。

QEMU 的优点是可以支持多种架构，可扩展，可移植，模拟速度快，可以在其他平台上运行 Linux 的程序，可以储存及还原运行状态。

同时，QEMU 也有部分缺点，比如对不常用的架构支持并不完善，比其他模拟软件难安装及使用，且除非使用 kvm 等加速器，否则其模拟速度仍不及 VMware 等其他虚拟软件。

## 2.2 QEMU 源码的编译及安装

为了更好地掌握 QEMU 的构造和原理，我们选择从命令行进行 QEMU 的源码编译及安装。从源码安装的另一个好处是能够避免 apt 仓库里 QEMU 因版本并非最新带来的未知 bug。

本次实验的安装环境是 VMware 上的 Ubuntu18.04 版本虚拟机，因此实际上是在进行“虚拟机套虚拟机”的二次虚拟化。

### 2.2.1 安装依赖包

```
##安装依赖
sudo apt-get install build-essential pkg-config zlib1g-dev
sudo apt-get install libglib2.0-0 libglib2.0-dev
sudo apt-get install libsdl1.2-dev
sudo apt-get install libpixmap-1-dev libfdt-dev
sudo apt-get install autoconf automake libtool
sudo apt-get install librbfd-dev
apt-get install libaio-dev
apt-get install flex bison
```

需要注意的是，本次使用的安装环境是 Ubuntu18.04 虚拟机，在其他版本的虚拟机上部分依赖包可能无法安装。

### 2.2.2 下载 QEMU 源码并解压.zip 压缩包

可以选择访问 QEMU 的官网 (<https://www.qemu.org/download/>) 进行压缩包的下载，也可以选择使用如下指令直接在 Linux 使用命令行下载及解压：

```
wget https://download.qemu.org/qemu-5.1.0.tar.xz
tar xvJf qemu-5.1.0.tar.xz
```

### 2.2.3 QEMU 的编译选项设置

从 Linux 命令行进入解压后的 qemu 文件夹，新建一个 build 文件并进入。

```
dicardo@ubuntu:~/Desktop/qemu$ mkdir build
dicardo@ubuntu:~/Desktop/qemu$ cd build
```

设置编译选项 ./configure，运行如下指令：

```
../configure --target-list=x86_64-softmmu --enable-kvm --enable-debug
```

编译选项的作用分别是：

```
--enable-kvm: 编译KVM模块，使QEMU可以使用KVM来访问硬件提供的虚拟化服务
--enable-vnc: 启用VNC（虚拟网络控制台的缩写）
--enable-werror: 编译时，将所有warning当作Error处理
--target-list: 选择目标机器的架构。默认是将所有的架构都编译，但指定之后可以更快地编译
```

```
dicardo@ubuntu:~/Desktop/qemu/build$ ./configure --target-list=x86_64-softmmu --enable-kvm --enable-debug
warn: ignoring non-existent submodule ui/keycodemapdb
warn: ignoring non-existent submodule tests/fp/berkeley-testfloat-3
warn: ignoring non-existent submodule tests/fp/berkeley-softfloat-3
warn: ignoring non-existent submodule meson
warn: ignoring non-existent submodule capstone
warn: ignoring non-existent submodule slirp
./scripts/git-submodule.sh: 78: ./scripts/git-submodule.sh: git: not found
./scripts/git-submodule.sh: failed to update modules

Unable to automatically checkout GIT submodules ''.
If you require use of an alternative GIT binary (for example to
enable use of a transparent proxy), then please specify it by
running configure by with the '--with-git' argument. e.g.

$ ./configure --with-git='tsocks git'
```

运行./configure 之后发现 SDL support 缺失（在一大串输出中间，显示为“no”）。接着安装 SDL2，执行如下指令：

```
sudo apt-get install libsdl2-2.0
sudo apt-get install libsdl2-dev
sudo apt-get install libsdl2-mixer-dev
sudo apt-get install libsdl2-image-dev
sudo apt-get install libsdl2-ttf-dev
sudo apt-get install libsdl2-gfx-dev
```

重新运行./configure 指令后，观察到 SDL support 选项设置为“yes”，效果图如下：

```
module support: NO
  host CPU: x86_64
  host endianness: little
  target list: x86_64-softmmu
  gprof enabled: NO
  sparse enabled: NO
  strip binaries: NO
  profiler: NO
  static build: YES
  SDL support: YES
  SDL image support: YES
  GTK support: NO
  GTK GL support: NO
  pixman: YES
  VTE support: NO
  TLS priority: "NORMAL"
  GNUTLS support: NO
  libcrypt: NO
```

## 2.2.4 编译及安装

运行 make 指令，等待编译完成。

```
dicardo@ubuntu:~/Desktop/qemu/build$ make
/usr/bin/python3 -B /home/dicardo/Desktop/qemu/meson/meson.py introspect --tests
--benchmarks | /usr/bin/python3 -B scripts/mtest2make.py > Makefile.mtest
./ninja2make -t ninja2make --omit clean dist uninstall cscope TAGS ctags < build.
ninja > Makefile.ninja
make[1]: 进入目录"/home/dicardo/Desktop/qemu/slirp"
GEN      /home/dicardo/Desktop/qemu/build/slirp/src/libslirp-version.h
CC       /home/dicardo/Desktop/qemu/build/slirp/src/tcp_timer.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/state.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/ip6_input.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/dhcpv6.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/ndp_table.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/ip_icmp.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/ip_input.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/slirp.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/vmstate.o
CC       /home/dicardo/Desktop/qemu/build/slirp/src/ip_output.o
```

接着运行 make install 指令，等待安装完成。

```
dicardo@ubuntu:~/Desktop/qemu$ make install
make: 对“install”无需做任何事。
dicardo@ubuntu:~/Desktop/qemu$ cd build
dicardo@ubuntu:~/Desktop/qemu/build$ make install
make[1]: 进入目录"/home/dicardo/Desktop/qemu/slirp"
make[1]: 对“all”无需做任何事。
make[1]: 离开目录"/home/dicardo/Desktop/qemu/slirp"
Generating qemu-version.h with a meson_exe.py custom command
/home/dicardo/Desktop/qemu/scripts/qemu-version.sh: 12: /home/dicardo/Desktop/qe
mu/scripts/qemu-version.sh: git: not found
Installing files.
Installing subdir /home/dicardo/Desktop/qemu/qga/run to /usr/local/var/run
Installation failed due to insufficient permissions.
Attempting to use polkit to gain elevated privileges...
Installing subdir /home/dicardo/Desktop/qemu/qga/run to /usr/local/var/run
Installing trace/trace-events-all to /usr/local/share/qemu
Installing qemu-system-x86_64 to /usr/local/bin
Installing qga/qemu-ga to /usr/local/bin
Installing qemu-keymap to /usr/local/bin
Installing qemu-img to /usr/local/bin
Installing qemu-io to /usr/local/bin
Installing qemu-nbd to /usr/local/bin
Installing storage-daemon/qemu-storage-daemon to /usr/local/bin
```

至此，我们已经完成了虚拟机上 QEMU 源码的编译及安装工作。下一步，我们开始通过命令行指令在 QEMU 上安装虚拟机。



## 2.3 QEMU 环境下安装并启动虚拟机

在完成 Linux 系统下 QEMU 的安装后，我们开始在 QEMU 上安装虚拟机。本次实验选择的是 Ubuntu20.04 版本的虚拟机。

### 2.3.1 进入相应文件夹

进入之前在 2.2.3 节设置编译选项之前创建的 build 文件夹，再进入 x86\_64-softmmu 子文件夹，通过以下命令查看文件目录：

```
dicardo@ubuntu:~/Desktop/qemu/build/x86_64-softmmu$ ls
config-target.mak  description-pak  qemu-system-x86_64
```

可以发现目录下除了 config-target.mak 的 makefile 文件和 description-pak 配置文件之外，还有一个没有后缀名的 qemu-system-x86\_64 可执行文件。可以通过以下指令来启动 qemu 上的虚拟机：

```
./qemu-system-x86_64
```

以上指令会启动默认镜像下的操作系统，如果想要指定虚拟机，只需要在该指令后面加上特定的镜像名称.img。

### 2.3.2 创建虚拟机镜像

可以使用如下指令创建一个指定大小和格式的镜像文件：

```
dicardo@ubuntu:~/Desktop/qemu/build/x86_64-softmmu$ qemu-img create -f qcow2 ubuntu.img 10G
Formatting 'ubuntu.img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=10737418240 lazy_refcounts=off refcount_bits=16
```

其中，-f 用于指定镜像格式，qcow2 是 QEMU 最常用的镜像格式，采用写时复制技术来优化性能。ubuntu.img 是镜像的名字。10G 是镜像文件的大小（此处实际上是虚拟机文件的最大可占用空间）。

然而，最初创建镜像文件时分配的最大镜像文件大小可能在后续的使用中不够用，因此可以使用如下指令重新分配镜像文件的大小（xxG 即重新分配的大小）：

```
qemu-img resize ubuntu.img xxG
```

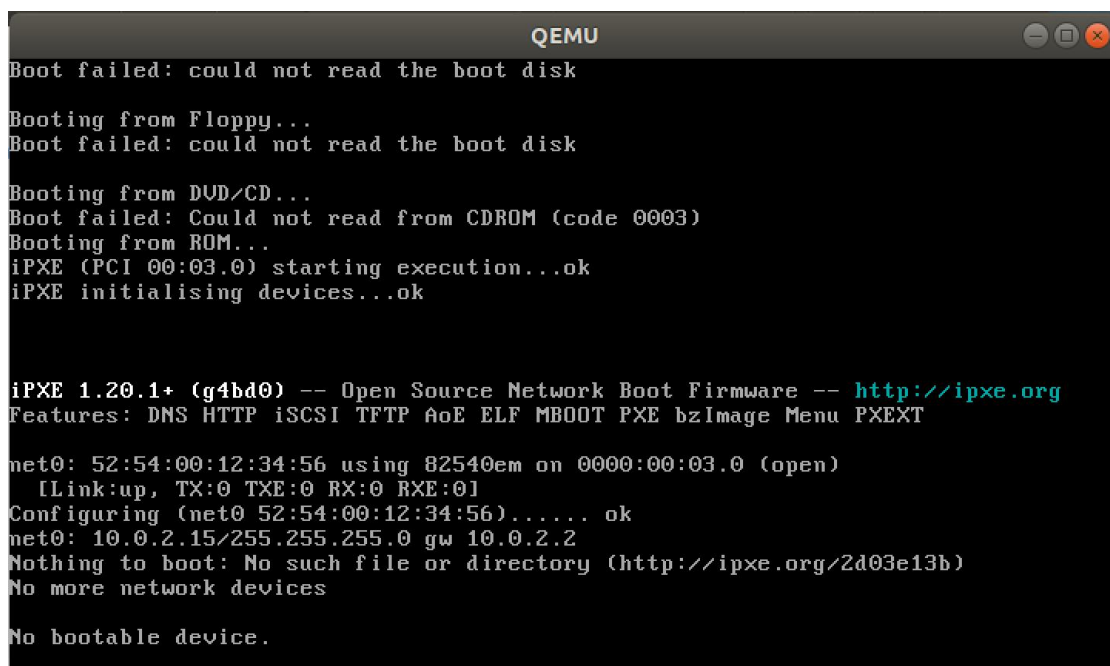
镜像文件创建完成之后，查看文件目录发现比之前多出来一个 ubuntu.img 的镜像文件。

```
dicardo@ubuntu:~/Desktop/qemu/build/x86_64-softmmu$ ls
config-target.mak  description-pak  qemu-system-x86_64  ubuntu.img
```

我们可以使用如下指令来启动 x86 架构的虚拟机：

```
./qemu-system-x86_64 ubuntu.img
```

此时会弹出窗口作为虚拟机的显示器。但因为 ubuntu.img 并未给虚拟机安装操作系统，故会显示 “No bootable device”。

A screenshot of a QEMU window titled "QEMU". The window shows a black terminal with white text. The text indicates a boot failure: "Boot failed: could not read the boot disk", "Booting from Floppy...", "Boot failed: could not read the boot disk", "Booting from DVD/CD...", "Boot failed: Could not read from CDRUM (code 0003)", "Booting from ROM...", "iPXE (PCI 00:03.0) starting execution...ok", "iPXE initialising devices...ok". Below this, it shows "iPXE 1.20.1+ (g4bd0) -- Open Source Network Boot Firmware -- http://ipxe.org" and a list of features: "Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT". It then shows network configuration: "net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)", "[Link:up, TX:0 TXE:0 RX:0 RXE:0]", "Configuring (net0 52:54:00:12:34:56)..... ok", "net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2", "Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)", "No more network devices", and finally "No bootable device." at the bottom.

```
QEMU
Boot failed: could not read the boot disk
Booting from Floppy...
Boot failed: could not read the boot disk
Booting from DVD/CD...
Boot failed: Could not read from CDRUM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.20.1+ (g4bd0) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
[Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)..... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
```

### 2.3.3 下载操作系统镜像文件

我们可以选择从 Ubuntu 官网 (<https://ubuntu.com/download/desktop>) 直接下载.iso 文件并复制粘贴到 2.3.1 节中提到的 x86\_64-softmmu 文件夹中;也可以选择从官网获取 Ubuntu20.04 版本的下载地址后，使用如下指令进行下载：

```
dicardo@ubuntu:~/Desktop/qemu/build/x86_64-softmmu$ wget https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso
--2020-09-22 19:19:59-- https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso
正在解析主机 releases.ubuntu.com (releases.ubuntu.com)... 91.189.88.247, 91.189.88.248, 91.189.91.124, ...
正在连接 releases.ubuntu.com (releases.ubuntu.com)|91.189.88.247|:443... 已连接。
已发出 HTTP 请求, 正在等待回应... 200 OK
长度: 2785017856 (2.6G) [application/x-iso9660-image]
正在保存至: "ubuntu-20.04.1-desktop-amd64.iso"

ubuntu-20.04.1-deskt  1%[          ] 31.09M  378KB/s  剩余 34m 59s
```

注意, 若过程中出现“段错误, 核心已转储”的错误提示信息:

```
sktop-amd64.iso      36%[=====>          ] 970.48M  1.88MB/s  剩余 16m 2s^
-desktop-amd64.iso  83%[=====          ] 2.18G  2.09MB/s  剩余 3m 47ss
                    99%[=====          ] 2.59G  2.39MB/s  剩余 0s
段错误 (核心已转储)
```

则使用如下命令从中断处继续下载:

```
wget -c https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso
```

直到出现“文件下载已完成; 不会进行任何操作”的提示信息。

```
dicardo@ubuntu:~/Desktop/qemu/build/x86_64-softmmu$ wget -c https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso
--2020-09-22 19:45:30-- https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-desktop-amd64.iso
正在解析主机 releases.ubuntu.com (releases.ubuntu.com)... 91.189.88.248, 91.189.91.124, 91.189.91.123, ...
正在连接 releases.ubuntu.com (releases.ubuntu.com)|91.189.88.248|:443... 已连接。
已发出 HTTP 请求, 正在等待回应... 416 Requested Range Not Satisfiable

文件已下载完成; 不会进行任何操作。
```

### 2.3.4 检查 KVM 模块是否可用

QEMU 使用 KVM 来提升虚拟机的性能, 如果不启用则会导致虚拟机性能的大幅损失。我们可以使用如下命令在母虚拟机上进行检测:

```
grep -E 'vmx|svm' /proc/cpuinfo
```

若有输出产生, 则代表硬件有虚拟化支持:



```

dicardo@ubuntu: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dicardo@ubuntu:~$ grep -E 'vmx|svm' /proc/cpuinfo
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon noptl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq vmx ss
se3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single
pti ssbd ibrs ibpb stibp tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_adjust bm
i1 avx2 smep bmi2 invpcid mpx rdseed adx snap clflushopt xsaveopt xsavec xsave
arat md_clear flush_l1d arch_capabilities
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon noptl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq vmx ss
se3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single
pti ssbd ibrs ibpb stibp tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_adjust bm
i1 avx2 smep bmi2 invpcid mpx rdseed adx snap clflushopt xsaveopt xsavec xsave

```

若没有输出，则关闭虚拟机，并在 VMware 中的设置 -> 处理器和内存 -> 高级选项中勾选开启虚拟化。



再次开启虚拟机并输入上述指令，即可观察到输出的产生。

其次，我们需要查看 KVM 模块是否已经加载，使用如下命令：

```
lsmod | grep kvm
```

如果出现如下显示，则说明 KVM 模块已经成功加载。

```
dicardo@ubuntu:~$ lsmod | grep kvm
kvm_intel          253952    0
kvm                655360    1 kvm_intel
```

最后，我们只需要保证在 2.2.3 节中设置编译选项时 enable 了 kvm 模块即可。

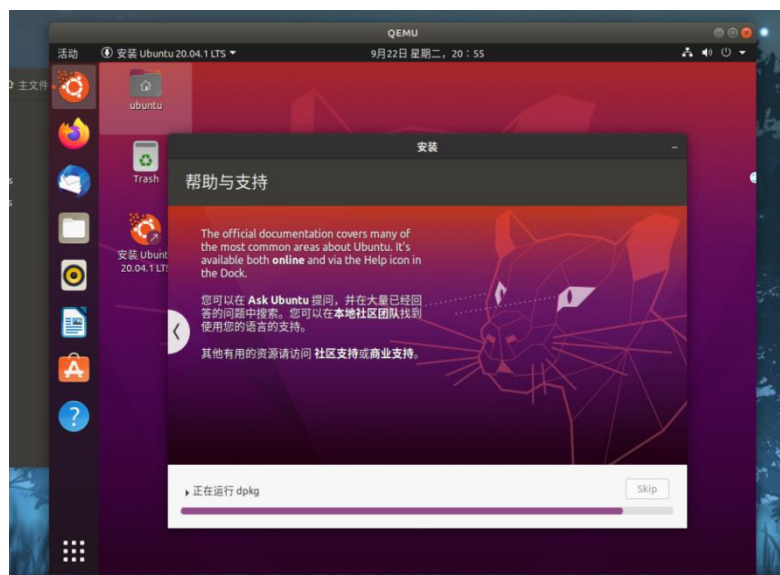
### 2.3.5 启动虚拟机安装操作系统

使用如下指令开始 ubuntu 虚拟机上操作系统的安装：

```
sudo ./qemu-system-x86_64 -m 4096 --enable-kvm ubuntu.img -cdrom ./ubuntu-20.04.1-desktop-amd64.iso
```

其中，**-m** 指定分配的虚拟机内存大小（默认单位是 MB）；**--enable-kvm** 使用 KVM 模块进行加速；**-cdrom** 添加 Ubuntu 的安装镜像。

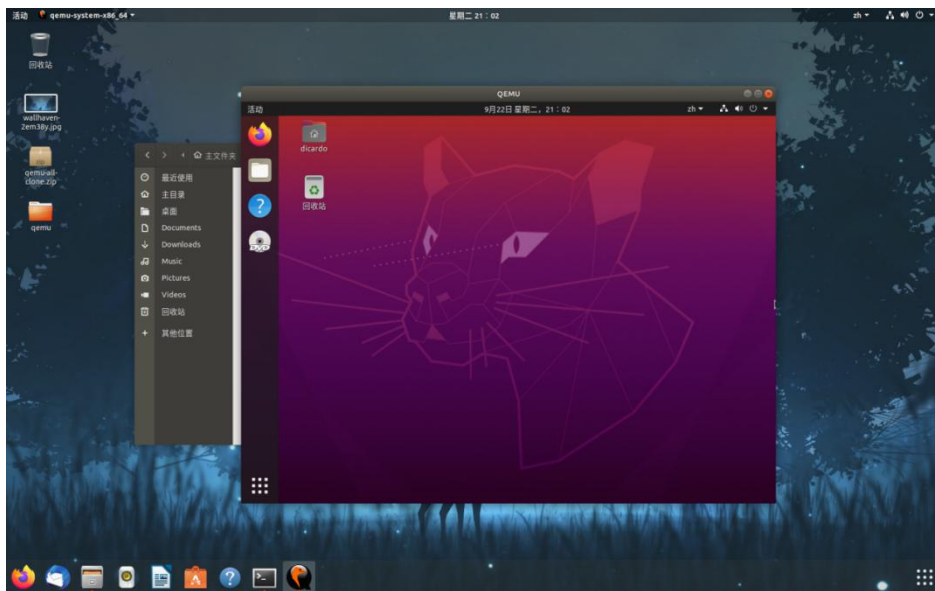
我们可以在弹出的窗口中操作虚拟机，并安装 Ubuntu20.04 的安装步骤安装操作系统。安装后重启虚拟机便可以从硬盘（ubuntu.img）启动。



之后要再启动虚拟机，只需要在命令行中执行如下指令：

```
sudo ./qemu-system-x86_64 -m 4096 -enable-kvm ubuntu.img
```

安装并配置好的 Ubuntu20.04 版本虚拟机图形界面如下所示:



## 2.4 QEMU 虚拟机性能测试及对比

在本节中, 我们利用 sysbench 这个 Linux 环境下的 benchmark 性能测试工具, 对虚拟机的 CPU、线程、磁盘 IO、内存和 Mutex 五个性能指标进行压力测试。Sysbench 是一个模块化、跨平台、多线程基准测试工具, 用于评估测试各种不同系统参数下的数据库负载情况。

在此基础上, 我们选择三个对象进行上述测试, 分别是母虚拟机 (Ubuntu18.04 版本), 启动 KVM 模块的 QEMU 虚拟机 (Ubuntu20.04 版本) 和未启动 KVM 模块的 QEMU 虚拟机 (Ubuntu20.04 版本), 并对三个对象下的几个性能指标进行横向比较, 同时得出两个对比结论:

(1) 母虚拟机 (在 VMware 上直接安装的一级虚拟机) 和 QEMU 虚拟机 (开启 KVM 模块的二级虚拟机) 之间的性能对比;

(2) QEMU 安装二级虚拟机时是否开启 KVM 模块对虚拟机性能的影响。

注意, 未开启 KVM 模块时, 虚拟机操作系统的安装和运行极度迟缓, 甚至无法加载出鼠标的光标。

### 2.4.1 Ubuntu 环境下 sysbench 工具的安装

首先，我们需要安装 sysbench 的依赖包：

```
sudo apt-get -y install make automake libtool pkg-config libaio-dev vim-common
```

Sysbench 工具可以直接通过以下命令从 Linux 的 apt 库内下载安装：

```
sudo apt-get install sysbench
```

可以通过命令行查看 sysbench 版本来验证工具是否安装成功：

```
dicardo@ubuntu:~$ sysbench --version  
sysbench 1.0.11
```

### 2.4.2 利用 sysbench 工具进行性能测试所需使用的指令

(1) CPU 测试：

```
sysbench --test=cpu --cpu-max-prime=2000 run
```

(2) 线程测试：

```
sysbench --test=threads --num-threads=500 --thread-yields=100 --thread-locks=4 run
```

(3) 磁盘 IO 测试：

a. 准备阶段：生成需要的测试文件，完成后会在当前目录下生成很多小文件。

```
sysbench --test=fileio --num-threads=16 --file-total-size=2G --file-test-mode=rndrw prepare
```

b. 运行阶段：

```
sysbench --test=fileio --num-threads=20 --file-total-size=2G --file-test-mode=rndrw run
```

c. 清理测试时生成的文件：

```
sysbench --test=fileio --num-threads=20 --file-total-size=2G --file-test-mode=rndrw cleanup
```

(4) 内存测试:

```
sysbench --test=memory --memory-block-size=8k --memory-total-size=1G run
```

(5) Mutex 测试:

```
sysbench --test=mux --num-threads=4 --mux-num=2000 --mux-locks=10000 --mux-loops=5000 run
```

需要注意的是,我们分别对上述三个测试对象进行性能测试时,为了控制变量,所使用的指令及参数均相同。接下来,我们会分点展示并分析每个测试对象的相应性能表现,及各虚拟机之间的横向比对。

### 2.4.3 CPU 性能测试

测试截图如下所示:

(1) 母虚拟机:

```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time

Prime numbers limit: 2000
Initializing worker threads...
Threads started!

CPU speed:
  events per second: 13401.17

General statistics:
  total time:          10.0001s
  total number of events: 134043

Latency (ms):
  min:                 0.07
  avg:                 0.07
  max:                 0.66
  95th percentile:    0.08
  sum:                 9976.50

Threads fairness:
  events (avg/stddev): 134043.0000/0.00
  execution time (avg/stddev): 9.9765/0.00
```



(2) 开启 KVM 的 QEMU 虚拟机:

```
CPU speed:
  events per second: 10497.41

General statistics:
  total time:                10.0002s
  total number of events:    104996

Latency (ms):
  min:                        0.07
  avg:                        0.09
  max:                        13.92
  95th percentile:          0.11
  sum:                        9966.91

Threads fairness:
  events (avg/stddev):       104996.0000/0.00
  execution time (avg/stddev): 9.9669/0.00
```

(3) 未开启 KVM 的 QEMU 虚拟机:

```
CPU speed:
  events per second:   400.44

General statistics:
  total time:                10.0082s
  total number of events:    4019

Latency (ms):
  min:                        1.05
  avg:                        2.45
  max:                        19.02
  95th percentile:          5.77
  sum:                        9850.71

Threads fairness:
  events (avg/stddev):       4019.0000/0.00
  execution time (avg/stddev): 9.8507/0.00
```

我们首先分析母虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 之间的性能差异。

从测试结果可以看到, S1 的 CPU 处理速度 (13401.14 event per second) 要快于 S2 (10497.41 event per second); 二者在 latency 方面的总时长大致相同, 但 S1 的最大 latency (0.66ms) 要远优于 S2 (13.92ms); 最后, 二者在 thread fairness 方面的事件数目上差别与处理速度的差别相同 (S1 为 134043.0000 / 0.00, S2 为 104996.0000 / 0.00), 而执行时间大致相同 (S1 为 9.9765 / 0.00, S2 为 9.9669 / 0.00)。

总结: S1 的整体性能表现要略优于 S2, 但开启了 KVM 后的 QEMU 虚拟机在各方面已经基本接近了在 VMware 上安装的虚拟机。在相同的测试时间内, 母虚拟机 S1 的 CPU 速度 (通过处理的事件数衡量) 要快于开启 KVM 的 QEMU 虚拟机 S2, 但相差在 30% 以内; Latency 方面, 二者平均延迟接近, 但 S1 的稳定性要优于 S2; Thread fairness 方面 S1 也要略优于 S2。

接下来, 我们分析开启 KVM 模块的 QEMU 虚拟机 (S2) 与未开启 KVM 模块的 QEMU 虚拟机 (S3) 之间的性能差异, 进而得出 KVM 模块对 QEMU 虚拟机性能提升的重要意义。

同上类似, 我们也分成 CPU 处理速度、latency 和 thread fairness 三个方面对二者进行评估。相较于 S2 的 10497.41 event per second, S3 的 CPU 处理速度会大幅度降低 (只有 400.44 event per second); Latency 方面, S2 的最大、最小、平均延迟都要显著优于 S3; Thread fairness 上, S2 在事件数目上 (104996.0000 / 0.00) 要明显优于 S3 (4019.0000 / 0.00), 但二者在执行时间上并没有明显差异。

总结: KVM 对 QEMU 虚拟机性能的提升有着至关重要的作用。在相同的测试时间内, 开启 KVM 模块的 QEMU 虚拟机无论是在 CPU 处理速度 (超出 95% 以上), latency (整体超出 90% 以上) 还是 thread fairness (整体超出 60% 左右), 都明显优于未开启 KVM 模块的 QEMU 虚拟机。KVM 模块的重要性可见一斑。

之后几个性能指标的分析方式也参考这样的分点分析。

#### 2.4.4 线程测试

测试截图如下所示:

(1) 母虚拟机:

```
Running the test with following options:
Number of threads: 500
Initializing random number generator from current time

Initializing worker threads...
Threads started!

General statistics:
  total time:                10.0278s
  total number of events:    188229

Latency (ms):
  min:                       0.05
  avg:                       26.60
  max:                       519.94
  95th percentile:          112.67
  sum:                       5007196.80

Threads fairness:
  events (avg/stddev):       376.4580/38.13
  execution time (avg/stddev): 10.0144/0.01
```

## (2) 开启 KVM 的 QEMU 虚拟机:

```

Running the test with following options:
Number of threads: 500
Initializing random number generator from current time

Initializing worker threads...

Threads started!

General statistics:
  total time:                10.3988s
  total number of events:    49184

Latency (ms):
  min:                       0.08
  avg:                       103.49
  max:                       1649.76
  95th percentile:          383.33
  sum:                       5089933.10

Threads fairness:
  events (avg/stddev):       98.3680/7.40
  execution time (avg/stddev): 10.1799/0.13

```

## (3) 未开启 KVM 的 QEMU 虚拟机:

```

General statistics:
  total time:                12.5536s
  total number of events:    1774

Latency (ms):
  min:                       0.37
  avg:                       3193.24
  max:                       9485.11
  95th percentile:          5918.87
  sum:                       5664812.30

Threads fairness:
  events (avg/stddev):       3.5480/0.89
  execution time (avg/stddev): 11.3296/0.77

```

首先分析母虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 在线程性能方面的差异。可以看到, 在总时间基本相同的前提下, S1 处理的事件数 (188229) 是 S2 (49184) 的三倍多; Latency 方面, 在最大、最小和平均延迟的整体表现上, S1 也比 S2 要短了三倍左右; Thread fairness 上, S1 的 events (avg / stddev) 是 376.4580 / 38.13, execution time (avg / stddev) 是 10.0144 / 0.01, 而 S2 的 events (avg / stddev) 是 98.3680 / 7.40, execution time (avg / stddev) 是 10.1799 / 0.13, 显然 S1 的线程 fairness 在整体上要明显优于 S2。

总结: 在线程性能上, 母虚拟机 S1 不论是在处理速度, latency 还是 thread fairness 上的表现, 都明显优于开启 KVM 模块的 QEMU 虚拟机 S2。S1 各项性能表现得分基本都是 S2 的三倍以上, 这个差距远大于在 CPU 性能上二者的差距, 说明 VMware 上安装的母虚拟机和开启 KVM 模块的 QEMU 虚拟机在线程性能上的差距还是非常明显的。

之后分析是否开启 KVM 模块对虚拟机线程性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到，即便 S3 的执行时间要略长于 S2，其执行事件数（1774）还是远少于 S2（49184）；Latency 方面，S2 的整体表现要远远优于 S3，平均延迟只有 S3 的十几分之一；Thread fairness 方面的差距也非常明显。

总结：在线程性能上，开启 KVM 模块的 QEMU 虚拟机 S2 要远远优于未开启 KVM 的 S3，且整体性能已经差出了十倍以上。这说明是否开启 KVM 模块对 QEMU 虚拟机的线程性能有着非常重要的影响。

#### 2.4.5 磁盘 IO 测试

测试截图如下所示：

(1) 母虚拟机：

```
2147483648 bytes written in 18.04 seconds (113.53 MiB/sec).
```

```
File operations:
  reads/s:          32091.67
  writes/s:         21395.28
  fsyncs/s:         68461.00

Throughput:
  read, MiB/s:      501.43
  written, MiB/s:   334.30

General statistics:
  total time:        10.0002s
  total number of events: 1220005

Latency (ms):
  min:               0.00
  avg:               0.16
  max:               21.10
  95th percentile:  1.04
  sum:               199320.94

Threads fairness:
  events (avg/stddev): 61000.2500/1525.51
  execution time (avg/stddev): 9.9660/0.00
```

(2) 开启 KVM 的 QEMU 虚拟机：

```
2147483648 bytes written in 41.39 seconds (49.49 MiB/sec).
```

```

File operations:
  reads/s:          898.81
  writes/s:         599.21
  fsyncs/s:         2155.07

Throughput:
  read, MiB/s:      14.04
  written, MiB/s:    9.36

General statistics:
  total time:        10.6130s
  total number of events: 36214

Latency (ms):
  min:               0.00
  avg:               5.52
  max:               47.39
  95th percentile:  17.01
  sum:               199760.41

Threads fairness:
  events (avg/stddev): 1810.7000/165.09
  execution time (avg/stddev): 9.9880/0.01

```

(3) 未开启 KVM 的 QEMU 虚拟机:

```
2147483648 bytes written in 127.67 seconds (16.04 MiB/sec).
```

```

  reads/s:          68.85
  writes/s:         46.91
  fsyncs/s:         299.70

Throughput:
  read, MiB/s:      1.08
  written, MiB/s:    0.73

General statistics:
  total time:        16.3725s
  total number of events: 4259

Latency (ms):
  min:               0.02
  avg:               46.84
  max:               692.99
  95th percentile:  139.85
  sum:               199504.04

Threads fairness:
  events (avg/stddev): 212.9500/18.50
  execution time (avg/stddev): 9.9752/0.04

```

首先分析母虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 在磁盘 IO 性能方面的差异。从创建相同数目测试文件的速度上来看, S1 为 113.53 MiB / sec, S2 为 49.49 MiB / sec。在读写速度, Throughput 和总事件数的整体表现上, S1 达到了惊人的 S2 的四十倍左右; Latency 方面, S1 在平均表现上是 S2 的几十分之一, 而最小和最大延迟接近; Thread fairness 上的表现也要远好于 S2。



总结: 在磁盘 IO 性能上, VMware 上安装的母虚拟机要远远好于开启 KVM 模块的 QEMU 虚拟机。其中原因一方面是后者属于二次虚拟化, 性能上本身会受较大影响; 另一方面是虚拟化之后对磁盘 IO 的变成了间接操作, 因此速度和性能上受到了明显的影响。

之后分析是否开启 KVM 模块对虚拟机磁盘 IO 性能的影响, 令开启 KVM 模块的为 S2, 未开启的为 S3。可以看到, 无论是读写速度、Throughput、总事件数、thread fairness 还是 latency, S2 的表现都要远远好于 S3。

总结: 在磁盘 IO 性能上, 开启 KVM 模块对 QEMU 虚拟机的性能提升十分明显。

#### 2.4.6 内存测试

测试截图如下所示:

(1) 母虚拟机:

```
Total operations: 131072 (1608862.04 per second)
1024.00 MiB transferred (12569.23 MiB/sec)

General statistics:
  total time:                0.0803s
  total number of events:    131072

Latency (ms):
  min:                       0.00
  avg:                       0.00
  max:                       0.25
  95th percentile:          0.00
  sum:                       65.77

Threads fairness:
  events (avg/stddev):       131072.0000/0.00
  execution time (avg/stddev): 0.0658/0.00
```

(2) 开启 KVM 的 QEMU 虚拟机:

```
Total operations: 131072 (886417.86 per second)
1024.00 MiB transferred (6925.14 MiB/sec)

General statistics:
  total time:                0.1465s
  total number of events:    131072

Latency (ms):
  min:                       0.00
  avg:                       0.00
  max:                       12.46
  95th percentile:          0.00
  sum:                       118.37

Threads fairness:
  events (avg/stddev):       131072.0000/0.00
  execution time (avg/stddev): 0.1184/0.00
```

(3) 未开启 KVM 的 QEMU 虚拟机:

```
Total operations: 131072 (90709.07 per second)
1024.00 MiB transferred (708.66 MiB/sec)

General statistics:
  total time:                1.4124s
  total number of events:    131072

Latency (ms):
  min:                        0.00
  avg:                        0.00
  max:                        9.02
  95th percentile:          0.00
  sum:                        639.36

Threads fairness:
  events (avg/stddev):       131072.0000/0.00
  execution time (avg/stddev): 0.6394/0.00
```

首先分析母虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 在内存性能方面的差异。可以看到, S1 的传输速率 (12569.23 MiB / sec) 大约为 S2 (6925.14 MiB / sec) 的两倍; 二者的最小、平均延迟都为 0, 但 S1 的最大延迟 (0.25ms) 仅为 S2 (12.46ms) 的几十分之一; Thread fairness 上二者也较为接近。

总结: 在内存性能上, VMware 上安装的母虚拟机要优于开启 KVM 模块的 QEMU 虚拟机, 但二者的差距并没有那么大。

之后分析是否开启 KVM 模块对虚拟机磁盘 IO 性能的影响, 令开启 KVM 模块的为 S2, 未开启的为 S3。可以看到 S2 的传输速率 (6925.14 MiB / sec) 大约为 S3 (708.66 MiB / sec) 的十倍; 二者的最小、平均延迟都为 0, 但 S2 的最大延迟 (12.46ms) 要略大于 S3 (9.02ms); Thread fairness 上二者仅在 execution time (avg / stddev) 上有不大的差异。

总结: 在内存性能上, 开启 KVM 的 QEMU 虚拟机在传输速率上大幅优于未开启 KVM 的, 但在 latency 方面却略有落后; 在 thread fairness 方面略有领先。

### 2.4.7 Mutex 测试

测试截图如下所示:

(1) 母虚拟机:

```

General statistics:
  total time:                0.0255s
  total number of events:    4

Latency (ms):
  min:                       18.45
  avg:                       21.77
  max:                       23.26
  95th percentile:          23.10
  sum:                       87.09

Threads fairness:
  events (avg/stddev):       1.0000/0.00
  execution time (avg/stddev): 0.0218/0.00

```

(2) 开启 KVM 的 QEMU 虚拟机:

```

General statistics:
  total time:                0.1245s
  total number of events:    4

Latency (ms):
  min:                       113.82
  avg:                       116.05
  max:                       120.38
  95th percentile:          121.08
  sum:                       464.19

Threads fairness:
  events (avg/stddev):       1.0000/0.00
  execution time (avg/stddev): 0.1160/0.00

```

(3) 未开启 KVM 的 QEMU 虚拟机:

```

General statistics:
  total time:                0.9866s
  total number of events:    4

Latency (ms):
  min:                       914.73
  avg:                       934.73
  max:                       954.37
  95th percentile:          960.30
  sum:                       3738.93

Threads fairness:
  events (avg/stddev):       1.0000/0.00
  execution time (avg/stddev): 0.9347/0.02

```

首先分析母虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 在 Mutex 性能方面的差异。可以看到, 在总事件数都为 4 的前提下, S1 的总时间 (0.0255s) 是 S2 (0.1245s) 的六分之一左右; Latency 上的整体表现 S1 略好于 S2。

总结: 在 Mutex 性能上, VMware 上安装的母虚拟机要优于开启 KVM 模块的 QEMU 虚拟机, 且在执行速率、延迟方面都要明显优于后者。

之后分析是否开启 KVM 模块对虚拟机 mutex 性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到，S2 的总时间 (0.1245s) 约为 S3 (0.9866s) 的八分之一；整体延迟也只为后者的八分之一；Thread fairness 上无明显差异。

总结：在 Mutex 性能上，开启 KVM 模块对处理速率及整体延迟的提升很大，但对 thread fairness 的提升有限。

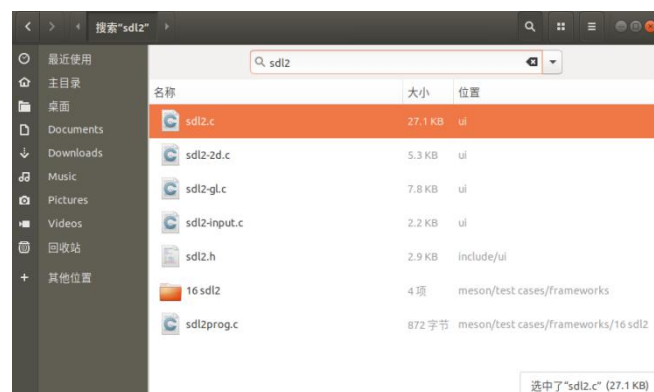
## 2.5 QEMU 部分源码修改

在完成了 QEMU 环境下虚拟机的安装、使用和测试之后，我们开始在能够通过编译的前提下，尝试修改 QEMU 的部分源码。

在经过对 QEMU 源码的分析之后，我们选择了“更改 QEMU 窗口标题”来作为源码修改的目标。选择的原因一方面是所需修改的代码量很小，适合作为入门学习者的我们进行操作；另一方面是因为其具有较高的实用性。当我们同时用 QEMU 开启多个虚拟机时，如果每个窗口的界面都是相同的，则在选择其中某个虚拟机时可能会出现不便甚至差错。在修改 QEMU 窗口栏标题之后，每个虚拟机的可辨识度会有一个较大的提升。同时，修改窗口标题也能较好的满足部分用户的个性化需求。

### 2.5.1 修改 QEMU 的部分代码

打开 qemu 文件夹下的 sdl2.c 文件：



定位到 `sdl_update_caption()` 函数，将以下部分代码：

```

150     if (qemu_name) {
151         snprintf(win_title, sizeof(win_title), "QEMU (%s-%d)%s", qemu_name,
152                 scon->idx, status);
153         snprintf(icon_title, sizeof(icon_title), "QEMU (%s)", qemu_name);
154     } else {
155         snprintf(win_title, sizeof(win_title), "QEMU%s", status);
156         snprintf(icon_title, sizeof(icon_title), "QEMU");
157     }
158
159     if (scon->real_window) {
160         SDL_SetWindowTitle(scon->real_window, win_title);
161     }

```

修改为:

```

150 //Modify the name of QEMU virtual machine on the UI windows here
151 if (qemu_name) {
152     snprintf(win_title, sizeof(win_title), "Chunyu Xue's QEMU (%s-%d)%s",
               qemu_name,
153               scon->idx, status);
154     snprintf(icon_title, sizeof(icon_title), "Chunyu Xue's QEMU (%s)",
               qemu_name);
155 } else {
156     snprintf(win_title, sizeof(win_title), "Chunyu Xue's QEMU%s", status);
157     snprintf(icon_title, sizeof(icon_title), "Chunyu Xue's QEMU");
158 }

```

### 2.5.2 重新创建未编译环境

为了在修改源码之后能够顺利编译, 我们首先需要删除本来已完成编译的环境删除。将本来编译好的 QEMU 环境利用 VMware 的 snapshot 功能进行快照备份, 完成后将 2.2.3 节创建的 build 文件夹删除, 重新创建并进入。运行如下指令进行编译:

```

../configure --target-list=x86_64-softmmu --enable-kvm --enable-debug
make

```

发现能够成功编译, 证明我们做的源码修改是可行的。

### 2.5.3 启动默认虚拟机镜像

进入 build 文件夹下的 x86\_64-softmmu 子文件夹, 用如下指令启动默认虚拟机镜像。窗口打开后, 观察到原来默认的标题“QEMU”被修改为“Chunyu Xue’s QEMU”。实验完成。

```

Chunyu Xue's QEMU
Boot failed: could not read the boot disk
Booting from Floppy...
Boot failed: could not read the boot disk
Booting from DUD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03:0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.20.1+ (q4hd0) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 02540em on 0000:00:03:0 (open)
[Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)..... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices
No bootable device.

```



### 3. 实验结论

本次实验中，我们成功完成了以下工作：

- (1) 从源码编译、安装 QEMU 模拟处理器；
- (2) 从命令行创建虚拟机映像，并在此基础上安装 Linux 操作系统；
- (3) 利用 sysbench 工具对母虚拟机、开启 KVM 的 QEMU 虚拟机和未开启 KVM 的 QEMU 虚拟机进行了一系列的性能测试，并进行了横、纵向的同时对比；
- (4) 对 QEMU 源码进行部分修改并重新编译安装，实现了 QEMU 窗口标题的自定义。

### 4. 实验心得

在刚刚开始本次实验时，我进行了大量的参考资料检索，但在数次尝试之后未果。每次在执行编译的时候都会出现各种各样的错误，有时是缺少某个依赖包，有时是环境变量未正确设置，甚至有时候出现错误的类型我都无从得知。

听取同学的建议后，我访问了 QEMU 的官网，并在官方教程中得到了不少灵感。在重新开始之后，我终于较为顺利地编译、安装好了 QEMU，并利用相关指令顺利完成了虚拟机的安装。

在设计性能测试时，虽然 sysbench 能够完全提供几项重要的性能测试，但考虑到如果只对开启了 KVM 模块的虚拟机进行测试会缺少一个对比组，难以通过数据（如 CPU 速度、延迟等）给出一个较为直观的结果，所以我设计了三组实验对象相互对照。这样，既能够比较得出开启 KVM 模块的 QEMU 虚拟机与母虚拟机之间的性能差距，也能够很直观地展示出 KVM 模块对 QEMU 虚拟机性能的重要性。

最后在考虑修改源码时，由于自己本身的能力有限，我只选择了一个较为简单，但非常实用的角度进行修改，即自定义 QEMU 窗口的标题。这个过程的参考资料在网上很少，因此我只能通过自己的不断摸索，不断尝试，最后成功完成了目标。

本次实验总体来说是较为成功的，我们较好地完成了所有实验目标，也增长了很多关于虚拟机编译、安装的知识，希望在下一次的实验中能够同样收获满满。