说明

以下文章基于ubuntu- XX 版本？ （请补充）

如果使用Gentoo Linux，可以参考 https://wiki.gentoo.org/wiki/RISC-V_Qemu_setup

（文章有点old）

创建工作目录

```
su && mkdir -p /workspace && mkdir -p linux-riscv && \
cd /workspace
```

如何编译内核

获取源码

为了获取具有完整commit记录和branch分支的linux源码，我们需要使用git工具。

安装git工具：

```
apt update && apt upgrade -y
apt install -y git
```

获取linux-kernel源码：

```
git clone https://github.com/torvalds/linux && cd linux
```

检查commit记录和分支：

```
git branch -a
git log
```

最小构建环境

具体最小构建环境版本要求请参考该文档：https://www.kernel.org/doc/html/v6.9-rc7/process/changes.html#changes

```
apt update && apt upgrade -y
apt install -y gcc clang rustc bindgen make flex bash  bison pahole mount
jfsutils reiserfsprogs xfsprogs  btrfs-progs pcmciautils quota ppp nfs-common
grub2-common udev python3-sphinx global build-essential libncurses-dev bison flex
libssl-dev libelf-dev bc gcc-riscv64-linux-gnu
```

配置与编译

在配置阶段，作为简单的切入点，可以选择默认配置 `defconfig` 或者使用当前内核的配置 `oldconfig`。

默认配置：根据机器型号选择合适的默认配置。

当前内核的配置：获取当前内核的配置作为编译的配置选项，若内核源码有新的选项，则会要求你一一确认。

其他配置选项参考文档：https://docs.kernel.org/admin-guide/README.html

```
make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- defconfig
```

编译直接使用make选项即可，`-j`选项后面的数字是使用核心数

```
make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j16
```

拷贝镜像文件到工作目录：

```
cp arch/riscv/boot/Image ../linux-riscv/
```

回到主目录：

```
cd ..
```

编译

在如何编译内核部分我们已经编译出来了内核镜像，但此时还缺少一个文件系统，我们这里选择使用`busybox`构建文件系统。

下载`busybox`源码：

```
git clone https://git.busybox.net/busybox && cd busybox
```

使用以下命令进行配置`busybox`，找到**Settings->Build static binary(no shared libs)**选项，按y键开启后退出：

```
make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- menuconfig
```

使用以下命令编译并进行安装`busybox`：

```
make ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- -j16 && make ARCH=riscv
CROSS_COMPILE=riscv64-linux-gnu- install
```

生成文件在`./_install/bin`目录下，查看busybox是否为riscv格式使用以下指令：

```
file ./_install/bin/busybox
```

如出现以下输出，则编译正确：

```
./_install/bin/busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float
ABI, version 1 (SYSV), statically linked,
BuildID[sha1]=461ff297e613b1624b8edb3ea22cf9474dfdeae7, for GNU/Linux 4.15.0,
stripped
```

拷贝`busybox`到工作目录：

```
cp _install/bin/busybox ../linux-riscv/
```

回到主目录：

```
cd ..
```

创建initramfs

由于我们并没有移植`uboot`去挂载文件系统，`qemu`的这个`bootloader`我并没有深入研究，所以这里打算使用`initramfs`去挂载文件系统。

使用如下命令构建`initramfs`工作目录：

```
cd linux-riscv && mkdir -p initramfs && cd initramfs && mkdir -p bin
```

移动`busybox`到bin目录：

```
mv ../busybox ./bin/
```

编写`init`进程文件，使用`busybox`作为脚本执行者：

```
vim init
# 将以下内容写入init文件
#!/bin/busybox sh
/bin/busybox mkdir -p /proc && /bin/busybox mount -t proc none /proc
/bin/busybox echo "Hello Linux"
/bin/busybox sh
```

赋予`init`文件可执行权限：

```
chmod +x init
```

回到工作目录创建`makefile`：

```
cd .. && vim Makefile
```

将以下内容写入Makefile：

```
.PHONY: initramfs

# 创建initramfs
# find . -print0: 这个命令使用find命令来查找当前目录(.)中的所有文件和子目录，并将它们的路径打印出来。-print0选项告诉find使用null字符(\0)来分隔文件名，这样可以确保文件名中的空格或特殊字符不会被误解
# cpio -ov --null --format=newc: cpio命令用于将文件归档成一个可供其他程序使用的归档文件。这里，-o选项表示创建归档文件，-v选项表示在处理文件时显示详细信息，--null选项告诉cpio使用null字符来分隔文件名，--format=newc选项表示使用新的c格式（newc）来归档文件。
# | gzip -9 > ../build/initramfs.cpio.gz: 这个管道符(|)将cpio命令的输出发送给gzip命令进行压缩。gzip -9表示使用最高的压缩级别进行压缩。最后，>符号将压缩后的数据写入build文件夹中的initramfs.cpio.gz文件。
initramfs:
        cd ./initramfs && mkdir -p ../build && find . -print0 | cpio -ov --null --format=newc | gzip -9 > ../build/initramfs.cpio.gz
```

```
# -M virt: 这个选项指定了使用virt平台（虚拟化硬件）进行模拟。
# -smp 4: 这个选项指定了虚拟机中的处理器数量为4个。
# -m 2G: 这个选项指定了虚拟机的内存大小为2GB。
# -display none: 这个选项指定了不显示图形界面。
# -serial stdio: 这个选项指定了将串行输出重定向到标准输入输出（stdio），这意味着虚拟机的输出将会显示在控制台上。
# -kernel ./Image: 这个选项指定了要加载的Linux内核文件的路径。
# -initrd ./build/initramfs.cpio.gz: 这个选项指定了要加载的初始RAM文件系统（initramfs）的路径。在虚拟机引导过程中，initramfs会被加载到内存中，提供启动所需的文件和程序。
# -append "root=/dev/ram": 这个选项指定了Linux内核的启动参数。在这里，root=/dev/ram表示将initramfs作为根文件系统加载到内存中。
run:
        qemu-system-riscv64     -M virt -smp 4 -m 2G \
                                -display none -serial stdio \
                                -kernel ./Image \
                                -initrd ./build/initramfs.cpio.gz \
                                -append "root=/dev/ram"
```

安装qemu

使用`apt`包管理器安装qemu：

```
apt install qemu-system-misc
```

运行测试

```
make initramfs && make run
```