实验0: Rinux环境搭建和内核编译

1 实验简介

搭建虚拟机、docker环境。通过在QEMU上运行Linux来熟悉如何从源代码开始将内核运行在QEMU模拟器上，并且掌握使用gdb协同QEMU进行联合调试，为后续实验打下基础。

2 实验环境

本次实验要求能够在虚拟机Linux系统中运行docker，需要安装：

- **VMware Workstation 15 Player** 或更新版本
- **Ubuntu 18.04.5 LTS**
- **docker**

下载oslab.tar

以下工具或软件(源码)包已在本实验所提供的Docker镜像中添加了，如果不使用DOcker可能还需要安装：

qemu 5.0.0

riscv-gnu-toolchain

buildroot 2020.08-rc1

linux 5.4.0-77-generic

riscv-pk

2.1 虚拟机环境

Linux受GNU通用公共许可证（GPL）保护，其内核源代码是完全开放的。现在很多Linux的网站都提供内核代码的下载。推荐你使用Linux的官方网站：http://www.kernel.org ，在这里你可以找到所有的内核版本。考虑到网络下载速度，我们也可以从https://mirrors.aliyun.com/linux-kernel/就近下载。

本实验将会在基于Linux内核开发的发行版操作系统Ubuntu中进行。请从https://mirrors.zju.edu.cn/ubuntu-releases/或者https://mirrors.aliyun.com/ubuntu-releases/下载ubuntu-18.04.5-desktop-amd64.iso。

使用VMware Workstation Player安装ubuntu 18.04.5 LTS虚拟机：

- 安装时设置虚拟机ubuntu的内存容量，推荐4GB，便于后续进行编译内核的实验；
- 安装时设置虚拟机ubuntu的硬盘容量，推荐不小于128GB；
- 设置虚拟机ubuntu的网络共享为NAT模式，使得虚拟机可以访问网络；

安装完毕并进入Linux系统后，更新软件源：

$ sudo apt-get update && sudo apt-get upgrade -y

安装VMWare Tools（高版本的环境，已经更名为open-vm-tools）：

$ sudo apt-get install open-vm-tools-desktop

$ sudo apt-get install open-vm-tools

安装gitee客户端（可选，也就是ubuntu环境中git的客户端），方便做实验：

$sudo apt install git-all

2.2下载实验docker镜像

我们的实验建议在Docker下，需要下载oslab.tar，下载链接在"学在浙大"课程网站上可以得到。
oslab.tar是容器镜像，包含了Linux系统的命令程序、QEMU环境、gcc环境、GDB、部分原代码等。

如果不使用Docker环境，还需要在你的Linux系统下配齐QEMU环境、gcc环境：

$ sudo apt-get install qemu

$ sudo apt-get install -y gcc make qemu build-essential module-assistant gcc-multilib g++-multilib

$ sudo dpkg --add-architecture i386

$ sudo apt-get install -y libc6:i386 libncurses5:i386 libstdc++6:i386

$ sudo apt-get install -y gdb-multiarch

3 实验基础知识介绍

3.1 Linux 使用基础

在Linux环境下，人们通常使用命令行接口来完成与计算机的交互。终端（Terminal）是用于处理该过程的一个应用程序，通过终端你可以运行各种程序以及在自己的计算机上处理文件。在类Unix的操作系统上，终端可以为你完成一切你所需要的操作。
下面我们仅对实验中涉及的一些概念进行介绍，你可以通过下面的链接来对命令行的使用进行学习：

1. The Missing Semester of Your CS Education >>Video<<
2. GNU/Linux Command-Line Tools Summary
3. Basics of UNIX

环境变量

当我们在终端输入命令时，终端会找到对应的程序来运行。我们可以通过`which`命令来做一些小的实验：

```
$ which gcc
/usr/bin/gcc
$ ls -l /usr/bin/gcc
lrwxrwxrwx 1 root root 5 5月  21  2019 /usr/bin/gcc -> gcc-7
```

可以看到，当我们在输入`gcc`命令时，终端实际执行的程序是`/usr/bin/gcc`。实际上，终端在执行命令时，会从`PATH`环境变量所包含的地址中查找对应的程序来执行。我们可以将`PATH`变量打印出来来检查一下其是否包含`/usr/bin`。

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:/home/phantom/.local/bin
```

在后面的实验中，如果你想直接访问`riscv64-unknown-linux-gnu-gcc`、`qemu-system-riscv64`等程序，那么你需要把他们所在的目录添加到目录中。

```
$ export PATH=$PATH:/opt/riscv/bin
```

可以在主目录的.bashrc文件的最后，添加上述命令行。然后，用如下命令验证其效果:

$ env | grep PATH -

3.2 Docker 使用基础

Docker 基本介绍

Docker 是一种利用容器（container）来进行创建、部署和运行应用的工具。Docker把一个应用程序运行需要的二进制文件、运行需要的库以及其他依赖文件打包为一个包（package），然后通过该包创建容器并运行，由此被打包的应用便成功运行在了Docker容器中。之所以要把应用程序打包，并以容器的方式运行，主要是因为在生产开发环境中，常常会遇到应用程序和系统环境变量以及一些依赖的库文件不匹配，导致应用无法正常运行的问题。Docker带来的好处是只要我们将应用程序打包完成（组装成为Docker imgae），在任意安装了Docker的机器上，都可以通过运行容器的方式来运行该应用程序，因而将依赖、环境变量等带来的应用部署问题解决了。
Docker和虚拟机功能上有共同点，但是和虚拟机不同，Docker不需要创建整个操作系统，只需要将应用程序的二进制和有关的依赖文件打包，因而容器内的应用程序实际上使用的是容器外Host的操作系统内核。这种共享内核的方式使得Docker的移植和启动非常的迅速，同时由于不需要创建新的OS，Docker对于容器物理资源的管理也更加的灵活，Docker用户可以根据需要动态的调整容器使用的计算资源（通过cgroups）。

Docker 安装

如果你在 Ubuntu 发行版上安装 Docker，安装使用方法请参见**4.1**。

3.3 QEMU 使用基础

什么是QEMU

QEMU最开始是由法国程序员Fabrice Bellard开发的模拟器。QEMU能够完成用户程序模拟和系统虚拟化模拟。用户程序模拟指的是QEMU能够将为一个平台编译的二进制文件运行在另一个不同的平台，如一个ARM指令集的二进制程序，通过QEMU的TCG（Tiny Code Generator）引擎的处理之后，ARM指令被转化为TCG中间代码，然后再转化为目标平台（比如Intel x86）的代码。系统虚拟化模拟指的是QEMU能够模拟一个完整的系统虚拟机，该虚拟机有自己的虚拟CPU，芯片组，虚拟内存以及各种虚拟外部设备，能够为虚拟机中运行的操作系统和应用软件呈现出与物理计算机完全一致的硬件视图。

如何使用 QEMU（常见参数介绍）

以下述命令为例，我们简单介绍QEMU的参数所代表的含义

```
$ qemu-system-riscv64 -nographic -machine virt -kernel
build/linux/arch/riscv/boot/Image  \
-device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0"   \
-bios default -drive file=rootfs.ext4,format=raw,id=hd0 \
-netdev user,id=net0 -device virtio-net-device,netdev=net0 -S -s
```

-nographic: 不使用图形窗口，使用命令行
-machine: 指定要emulate的机器，可以通过命令`qemu-system-riscv64 -machine help`查看可选择的机器选项
-kernel: 指定内核image
-append cmdline: 使用cmdline作为内核的命令行
-device: 指定要模拟的设备，可以通过命令`qemu-system-riscv64 -device help`查看可选择的设备，通过命令`qemu-system-riscv64 -device <具体的设备>,help`查看某个设备的命令选项
-drive, file=<file_name>: 使用'file'作为文件系统
-netdev user,id=str: 指定user mode的虚拟网卡, 指定ID为str
-S: 启动时暂停CPU执行(使用'c'启动执行)
-s: -gdb tcp::1234 的简写
-bios default: 使用默认的OpenSBI firmware作为bootloader

更多参数信息可以参考这里

3.4 GDB 使用基础

什么是 GDB

GNU调试器（英语：GNU Debugger，缩写：gdb）是一个由GNU开源组织发布的、UNIX/LINUX操作系统下的、基于命令行的、功能强大的程序调试工具。借助调试器，我们能够查看另一个程序在执行时实际在做什么（比如访问哪些内存、寄存器），在其他程序崩溃的时候可以比较快速地了解导致程序崩溃的原因。
被调试的程序可以是和gdb在同一台机器上（本地调试，or native debug），也可以是不同机器上（远程调试， or remote debug）。

总的来说，gdb可以有以下4个功能

- 启动程序，并指定可能影响其行为的所有内容
- 使程序在指定条件下停止
- 检查程序停止时发生了什么
- 更改程序中的内容，以便纠正一个bug的影响

GDB 基本命令介绍

(gdb) start：单步执行，运行程序，停在第一执行语句
(gdb) next：单步调试（逐过程，函数直接执行）,简写n
(gdb) run：重新开始运行文件（run-text：加载文本文件，run-bin：加载二进制文件），简写r
(gdb) backtrace：查看函数的调用的栈帧和层级关系，简写bt
(gdb) break 设置断点。比如断在具体的函数就break func；断在某一行break filename:num
(gdb) finish：结束当前函数，返回到函数调用点
(gdb) frame：切换函数的栈帧，简写f
(gdb) print：打印值及地址，简写p
(gdb) info：查看函数内部局部变量的数值，简写i；查看寄存器的值i register xxx
(gdb) display：追踪查看具体变量值

更多命令可以参考100个gdb小技巧

3.5 LINUX 内核编译基础

交叉编译

交叉编译指的是在一个平台上编译可以在另一个平台运行的程序，例如在x86机器上编译可以在arm平台运行的程序，交叉编译需要交叉编译工具链的支持。

内核配置

内核配置是用于配置是否启用内核的各项特性，内核会提供一个名为 `defconfig`(即default configuration) 的默认配置，该配置文件位于各个架构目录的 `configs` 文件夹下，例如对于RISC-V而言，其默认配置文件为 `arch/riscv/configs/defconfig`。使用 `make ARCH=riscv defconfig` 命令可以在内核根目录下生成一个名为 `.config` 的文件，包含了内核完整的配置，内核在编译时会根据 `.config` 进行编译。配置之间存在相互的依赖关系，直接修改defconfig文件或者 `.config` 有时候并不能达到想要的效果。因此如果需要修改配置一般采用 `make ARCH=riscv menuconfig` 的方式对内核进行配置。

常见参数

ARCH 指定架构，可选的值包括arch目录下的文件夹名，如x86,arm,arm64等，不同于arm和arm64，32位和64位的RISC-V共用 `arch/riscv` 目录，通过使用不同的config可以编译32位或64位的内核。

CROSS_COMPILE 指定使用的交叉编译工具链，例如指定 `CROSS_COMPILE=riscv64-unknown-linux-gnu-`，则编译时会采用 `riscv64-unknown-linux-gnu-gcc` 作为编译器，编译后可以在RISC-V平台上运行的kernel。

CC 指定编译器，通常指定该变量是为了使用clang编译而不是用gcc编译，Linux内核在逐步提供对clang编译的支持，arm64、x86、RISC-V等已经能够很好的使用clang进行编译。

常用编译选项，我们将在后续实验的Makefile中频繁遇到

4 实验步骤

通常在Linux环境下，`$` 提示符表示当前运行的用户为普通用户，`#` 代表当前运行的用户为特权用户。

但注意，在下文的示例中：

- 以 `###` 开头的行代表注释，
- 以 `$` 开头的行代表在你的宿主机/虚拟机上运行的命令，

- 以`#`开头的行代表在`docker`中运行的命令，
- 以`(gdb)`开头的行代表在`gdb`中运行的命令。

> **在执行每一条命令前，请你对将要进行的操作进行思考，给出的命令不需要全部执行，并且不是所有的命令都可以无条件执行，请不要直接复制粘贴命令去执行。**

4.1 搭建docker环境

```
###请注意，你的系统可能不需要一定按照下面的顺序执行命令。

### docker安装可使用下面命令。安装并运用命令行工具curl(client URL)，与docker的Web服务器交互。
$ sudo apt-get install curl
$ curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun

### 将用户加入docker组，免sudo
$ sudo usermod -aG docker $USER   ### 注销后重新登陆生效。Linux关机，重新启动后，生效。
### 使用官方安装方法还需执行以下语句
### sudo chmod a+rw /home/$USER/.docker/config.json

### 将oslab.tar文件，复制到主目录

### 导入docker镜像
$ cat oslab.tar | docker import - oslab:2021
### 执行命令后若出现以下错误提示
### ERROR: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock
### 可以使用下面命令为该文件添加权限来解决
### $ sudo chmod a+rw /var/run/docker.sock

### 查看docker镜像
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
oslab               2021                d7046ea68221        5 seconds ago
2.89GB

### 从镜像创建一个容器
$ docker run -it oslab:2021 /bin/bash    ### -i:交互式操作 -t:终端
root@368c4cc44221:/#                     ### 提示符变为 '#' 表明成功进入容器
                                         ### 前部的字符串根据容器而生成，为容器id
root@368c4cc44221:/# exit（或者CTRL+D）   ### 从容器中退出。此时运行docker ps，运行容器的列表为空

### 查看当前运行的容器
$ docker ps
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
### 查看所有存在的容器
$ docker ps -a
CONTAINER ID  IMAGE        COMMAND      CREATED         STATUS
PORTS NAMES
368c4cc44221  oslab:2020  "/bin/bash"  54 seconds ago Exited(0) 30 seconds ago
agnesi
```

```
### 启动处于停止状态的容器
$ docker start 368c       ### 368c 为容器id的前四位，id开头的几位便可标识一个容器，你的计算机可能不是这个id
$ docker ps              ### 可看到容器已经启动
CONTAINER ID  IMAGE       COMMAND      CREATED              STATUS
PORTS NAMES
368c4cc44221  oslab:2021  "/bin/bash"  About a minute ago  Up 16 seconds
agnesi

### 进入已经运行的容器 oslab的密码为2020
$ docker exec -it -u oslab -w /home/oslab 36 /bin/bash
oslab@368c4cc44221:~$    ###在容器中了

###可能会用到的命令
### 挂载本地目录，如：
### $ docker run -it -v /home/lab:/home/lab0 oslab:2021 /bin/bash
### docker与本地文件复制命令，如：
### $ docker cp /home/lab 368c4cc44221:/home/lab0
### 使用docker的帮助信息
### $ docker COMMAND --help

### 进入docker后，按4.2-4.5指导进行下一步实验
```

4.2 编译 linux 内核

```
### 进入实验目录并设置环境变量
# pwd
/home/oslab
# cd lab0
# export TOP=`pwd`
# export RISCV=/opt/riscv
# export PATH=$PATH:$RISCV/bin

# mkdir -p build/linux
# make -C linux O=$TOP/build/linux   CROSS_COMPILE=riscv64-unknown-linux-gnu-
ARCH=riscv CONFIG_DEBUG_INFO=y defconfig all -j$(nproc)
```

若在编译内核时得到下面的报错信息，请增加虚拟机内存(从2GB增至4GB)。

```
...
LD  vmlinux.o
Killed
.../Makefile:1135: recipe for target 'vmlinux' failed
make[2]: *** [vmlinux] Error 137
```

4.3 使用QEMU运行内核

```
### 用户名root，没有密码
# qemu-system-riscv64 -nographic -machine virt -kernel
build/linux/arch/riscv/boot/Image  \
 -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0"
\
 -bios default -drive file=rootfs.ext4,format=raw,id=hd0 \
 -netdev user,id=net0 -device virtio-net-device,netdev=net0

退出QEMU模拟器的方法为：使用`ctrl`+`a`(macOS下为`control+a`)，松开后再按下`x`键即可退出
qemu
```

4.4 使用 gdb 对内核进行调试

```
### Terminal 1
# qemu-system-riscv64 -nographic -machine virt -kernel
build/linux/arch/riscv/boot/Image \
 -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0"
\
 -bios default -drive file=rootfs.ext4,format=raw,id=hd0 \
 -netdev user,id=net0 -device virtio-net-device,netdev=net0 -S -s

### Terminal 2
# riscv64-unknown-linux-gnu-gdb build/linux/vmlinux
(gdb) target remote localhost:1234  ### 连接 qemu
(gdb) b start_kernel                ### 设置断点
(gdb) continue                      ### 继续执行
(gdb) quit                          ### 退出 gdb
```

5 实验任务与要求

- 请各位同学独立完成实验，任何抄袭行为都将使本次实验判为0分。
- 编译内核并用 gdb + QEMU 调试，在内核初始化过程中（用户登录之前）设置断点，对内核的启动过程进行跟踪，并尝试使用gdb的各项命令（如backtrace、finish、frame、info、break、display、next等）。
- 在学在浙大中提交实验报告，记录实验过程并截图（4.1-4.4），对每一步的命令以及结果进行必要的解释，记录遇到的问题和心得体会。

本文内容基于如下同学的贡献而改编：

> 张文龙
> 周侠 (gdb qemu riscv-toolchain)
> 管章辉 (docker)
> 徐金焱 (gdb qemu riscv-toolchain)
> 刘强 孙家栋 周天昱

浙江大学实验报告

课程名称： 操作系统

实验项目名称：

学生姓名： 学号：

电子邮件地址：

实验日期： 年 月 日

一、实验内容

记录实验过程并截图，对每一步的命令以及结果进行必要的解释

三、讨论、心得（20分）

在这里写：实验过程中遇到的问题及解决的方法，你做本实验体会