本科生实验报告

实验课程: 操作系统原理实验

任课教师: 刘宁

实验题目:编译内核/使用现有内核实现操作系统

专业名称: 计算机科学与技术

学生姓名:罗弘杰

学生学号: 22336173

实验地点: 实验中心 D503

实验时间: 2024/3/3

Section 1 实验概述

• 熟悉现有 Linux 内核的编译过程和启动过程，并在自行编译内核的基础上构建简单应用并启动；利用 精简的 Busybox 工具集构建简单的 OS，熟悉现代操作系统的构建过程。此外，熟悉编译环境、相关工具集，并能够实现内核远程调试；

1. 独立完成实验 5 个部份环境配置、编译 Linux 内核、Qemu 启动内核并开启远程调试、制作 Initramfs 和 编译并启动 Busybox。

2. 编写实验报告、结合实验过程来谈谈你完成实验的思路和结果，最后需要提供实验的 5 个部份的程序 运行截屏来证明你完成了实验。

3. 实验不限语言， C/C++/Rust 都可以。

4. 实验不限平台， Windows、Linux 和 MacOS 等都可以。

5. 实验不限 CPU， ARM/Intel/Risc-V 都可以。。

Section 2 预备知识与实验环境

该节总结实验需要用到的基本知识，以及主机型号、代码编译工具、重要三方库的版本号信息等。

- 预备知识：x86 汇编语言程序设计、Linux 系统命令行工具
- 实验环境：
    - 虚拟机版本/处理器型号：ubuntu-18.0.4 , 阿里云服务器通用 cpu
    - 代码编辑环境： vim
    - 代码编译工具： gcc
    - 重要三方库信息：无

Section 3 实验任务

该节描述需要完成的几个实验任务，即重述实验题目的总要求，建议使用项目编号分点阐述。详细要求可在下一节【实验步骤与实验结果】中列出。

- 实验任务 1：完成虚拟机搭建，配置操作系统实验需要的环境，包括 c 语言编译器，链接器，qemu，busybox,下载操作系统内核等等。
- 实验任务 2： 编译并启动 Kernel
- 实验任务 3：编写 initramfs,加载到操作系统

- 实验任务 4：制作并编译启动 Busybox

Section 4 实验步骤与实验结果

该节描述每个实验任务的具体的完成过程，包括思路分析、代码实现与执行、结果展示三个部分，实验任务之间的划分应当清晰明了，实验思路分析做到有逻辑、有条理。

---------- 实验任务 1 ----------

- 任务要求：配置虚拟机需要的环境，包括 c 语言编译环境，qemu 虚拟机程序等等。
- 思路分析：主要分析如何得到预期结果，有哪些要注意的地方等。
- 实验步骤：给出关键部分的代码实现/命令行的执行过程，辅以必要的文字描述（可从指导书上摘抄）。

命令行与代码块可以通过特殊背景进行区分，或者添加文本边框，建议在 IDE 或者 gedit 中编辑好后再复制/粘贴，可以选出与实验任务直接相关的关键部分进行粘贴，不要整段复制，代码英文字体建议采用 Consolas 五号字。

```
    sudo apt install nasm  # nasm: 是一个汇编语言编译器用于编译汇编语言程序
    sudo apt install qemu  # qemu: 是一个虚拟化和仿真软件，可以模拟多种硬件平台，并在其上运行不同的操作系统。
    sudo apt install cmake #是一个跨平台的项目构建工具，它使用简单的配置文件来控制项目的构建过程。
    sudo apt install libncurses5-dev #是用于开发基于文本的用户界面（TUI）程序的库文件。
    sudo apt install bison #是一个用于生成解析器和编译器的工具。
    sudo apt install flex  #是一个用于生成词法分析器的工具
    sudo apt install libssl-dev #是用于开发使用 OpenSSL 加密库的应用程序的库文件。
    sudo apt install libc6-dev-i386 #是一个用于开发 32 位应用程序的库文件。
```

下面是一个汇编程序实现过程调用的案例：

```
    call read_disk
    jmp 0x0000:0x7e00
    jmp $
    read_disk:
        mov ax, 0
        mov es, ax
        mov bx, 0x7e00
        mov ah, 0x02
        mov al, 0x05
```

```
times 510-($-$$) db 0
db 0x55, 0xaa
```

上面是深色背景的参考样式（直接从 VSCode 中复制得到），也可以采用下面的基于文字边框的浅色样式：

```
call read_disk
jmp 0x0000:0x7e00
jmp $
read_disk:
    mov ax, 0
    mov es, ax
    mov bx, 0x7e00
    mov ah, 0x02
    mov al, 0x05
times 510-($-$$) db 0
db 0x55, 0xaa
```

- 实验结果展示：
- 

---------- 实验任务 2 ----------

- 任务要求：下载 linux-5.10.212 的内核 kernel, 编译，并在 qemu 平台上进行操作系统启动，使用 gdb 远程调试。
- 思路分析：首先要从 kernel.org 上下载官方的 linux 内核，然后进行编译，在

配置文件中选择运行远程调试，然后使用 gdb 连接 qemu 端口进行调试。

- 实验步骤：
-

```
wget http……linux-5.10.212.tar.bz2 #从官网上下载内核打包文件
xz -d linux-5.10.212.tar.xz
tar -xvf linux-5.10.212.tar
cd linux-5.10.212 #解压内核文件并进入其文件夹
```

make i386_defconfig　#这个命令用于生成一个名为 i386_defconfig 的配置文件。配置文件包含了构建内核所需的各种选项和参数。i386_defconfig 是一个预定义的配置，适用于 x86 架构的 32 位系统。通过运行这个命令，你将使用默认的配置选项来构建内核。如果你想自定义配置，可以在运行 make menuconfig 之前编辑这个配置文件。
make menuconfig　#这个命令打开了一个文本界面的配置工具，允许你交互式地选择内核配置选项。你可以在这里启用或禁用特定的功能、驱动程序、文件系统等。使用箭头键浏览选项，空格键选择或取消选择，然后保存并退出。这是一个强大的工具，允许你根据你的需求定制内核。
make -j8 #使用 8 个核来编译内核
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.169/arch/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic
‘’’

- qemu-system-i386：这是 QEMU 的命令行工具，用于启动虚拟机。
- -kernel linux-5.10.169/arch/x86/boot/bzImage：指定了 Linux 内核镜像的路径，即要启动的内核文件。
- -s：启用了 **GDB 服务器**，它监听在端口 1234 上，允许你连接调试器（如 GDB）来调试虚拟机。
- -S：在启动时暂停虚拟机的执行，等待调试器连接。这对于在虚拟机开始运行之前设置断点很有用。
- -append "console=ttyS0"：追加了指定的内核命令行参数。在这里，它设置了控制台使用串口（ttyS0）。
- -nographic：禁用图形界面，使虚拟机只能通过命令行访问。

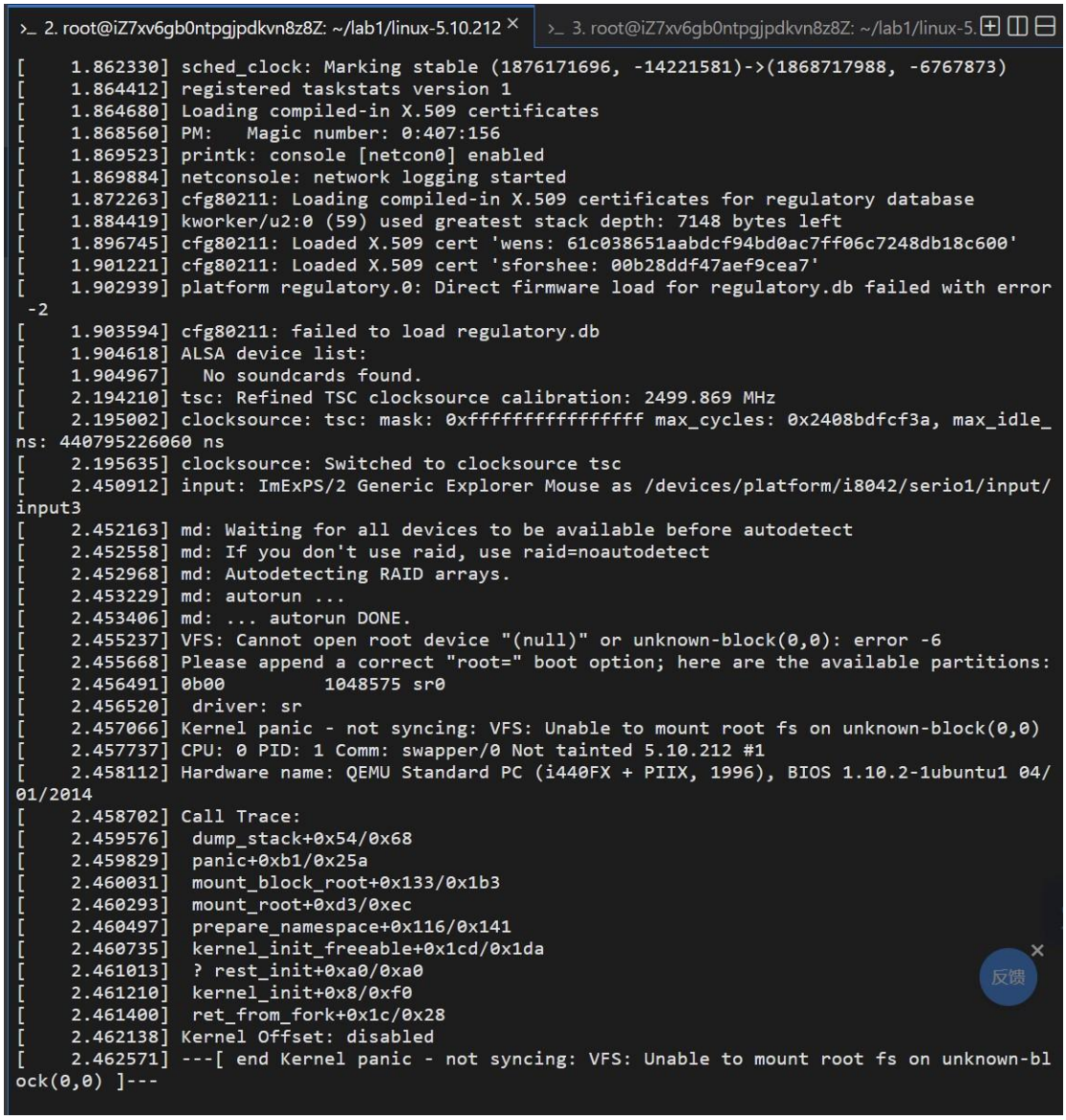
‘’’’
新开一个终端，然后使用 gdb 来调试
cd lab1/linux-5.10.212
gdb
file linux-5.10.212/vmlinux　#加载符号表 ，在调试和分析 Linux 内核时，加载符号表非常有用。符号表将内核代码中的函数名、变量名与实际的机器码关联起来。这样，调试器就能更好地理解内核代码，帮助你定位问题。 vmlinux 和 vmlinuz：vmlinux 是未压缩的内核镜像，用于调试和分析。而 vmlinuz 是压缩过的可引导内核镜像，用于实际引导操作系统。
target remote:1234 #gdb 连接远程的 qemu 调试端口
break start_kernel #在 start-kernel 函数建立断点，start_kernel：这个函数是 Linux 内核的入口点。当你启动 Linux 操作系统时，首先会执行这个函数。它负责初始化内核的各个子系统、驱动程序、数据结构等。具体来说，start_kernel 的主要任务包括：
1，初始化内核的基本数据结构，如进程调度、内存管理、文件系统等。
2，初始化中断处理和定时器。
3，加载内核模块。
4，启动初始化进程（通常是 init 进程）。

●

- 通过执行前述代码，可得下图结果。

展示：看到 call trace 打出来的是在 initrd_load 的时候出错，原因很简单，因为启动系统 的时候只指定了 bzImage，没有指定 initrd 文件，系统无法 mount 上 initrd (init ram disk) 及其 initramfs 文件 系统。

:



---------- 实验任务 3 ----------

- 任务要求：制作 Initramfs
- 思路分析：在前面调试内核中，我们已经准备了一个 Linux 启动环境，但是缺少 initramfs。我们可以做一个最简单的 Hello World initramfs，来直观地理解 initramfs。

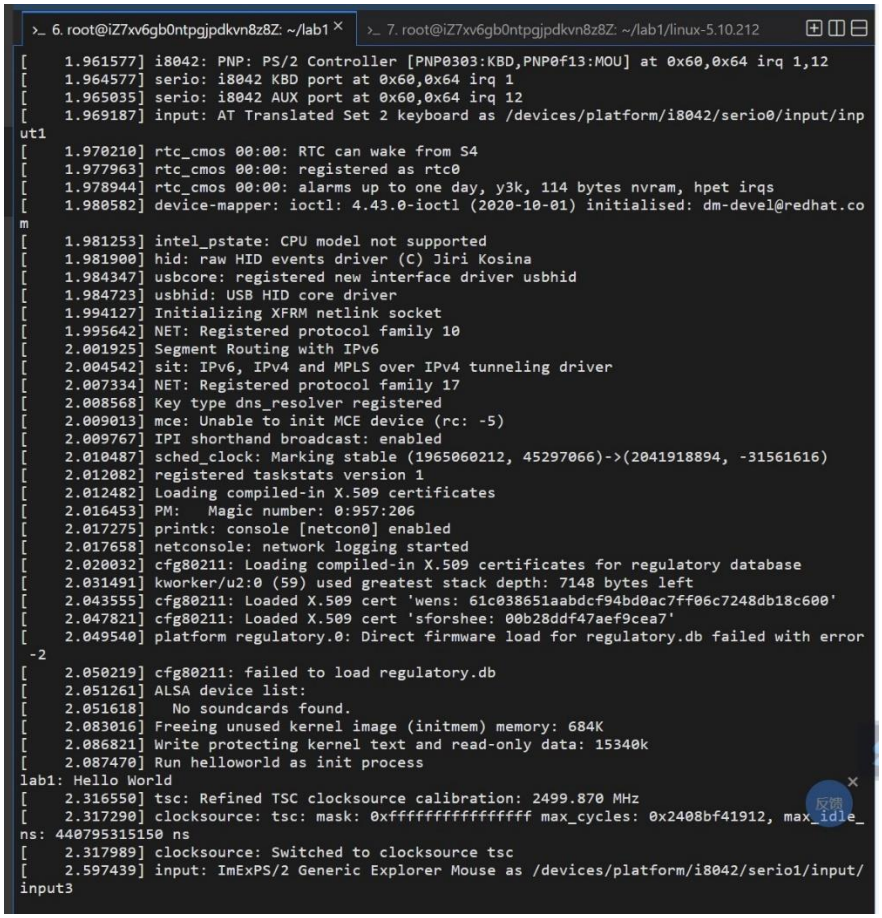
- 实验步骤：。

Vim
#inc
{ pri
fflus
whil

gcc -o helloworld -m32 -static helloworld.c #编译为 32 位可执行文件
echo helloworld | cpio -o --format=newc > hwinitramfs #命令将字符串 “helloworld” 写入一个名为 hwinitramfs 的文件中。这个文件使用 newc 格式，这是一种用于 initramfs 的格式。initramfs 是一个临时的根文件系统，用于在 Linux 启动过程中加载必要的模块和文件。您的命令将创建一个包含 “helloworld” 的 initramfs 文件，以备用于您之前提到的 qemu-system-i386 命令中。这样，当您启动虚拟机时，它将使用这个 initramfs 文件作为临时根文件系统。

qemu-system-i386 -kernel linux-5.10.169/arch/x86/boot/bzImage -initrd hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -nographic # -initrd hwinitramfs：这是一个初始化 RAM 文件系统（initramfs），它在启动过程中作为临时根文件系统使用。hwinitramfs 是您提供的 initramfs 文件的路径。
重复 gdb 调试过程

-
- 实验结果展示：通过执行前述代码，可得下图结果：
-



-

---------- **实验任务 4** ----------

- 任务要求：编译启动 Busybox
- 思路分析：主要分析如何得到预期结果，有哪些要注意的地方等。

- 实验步骤：

Wget http……busybox-1.33.0.tar.xz
tar -xf Busybox_1_33_0.tar.gz
make defconfig
make menuconfig
进入 settings，然后在 Build BusyBox as a static binary(no shared libs)处输入 Y 勾选，然后分别设置()Additional CFLAGS 和() Additional LDFLAGS 为(-m32 -march=i386) Additional CFLAGS 和(-m32) Additional
LDFLAGS。保存退出，
make -j8 #编译 Busybox
make install
制作 Initramfs • 将安装在_install 目录下的文件和目录取出放在~/lab1/mybusybox 处:
cd ~/lab1
mkdir mybusybox
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}} # mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}：这个命令会创建一系列目录，包括 bin、sbin、etc、proc、sys 以及 usr/bin 和 usr/sbin。其中，-p 选项确保如果父目录不存在，也会创建它们，而 -v 选项则会显示每个创建的目录。

cp -av busybox-1_33_0/_install/* mybusybox/ #这个命令会将 BusyBox 安装目录_install 中的文件和目录复制到您自定义的 mybusybox 目录中。其中，-a 选项会保留文件的属性（如权限和时间戳），而 -v 选项会显示详细信息。
cd mybusybox
initramfs 需要一个 init 程序，可以写一个简单的 shell 脚本作为 init。用 vim 打开文件 init，复制入如 下内容，保存退出。

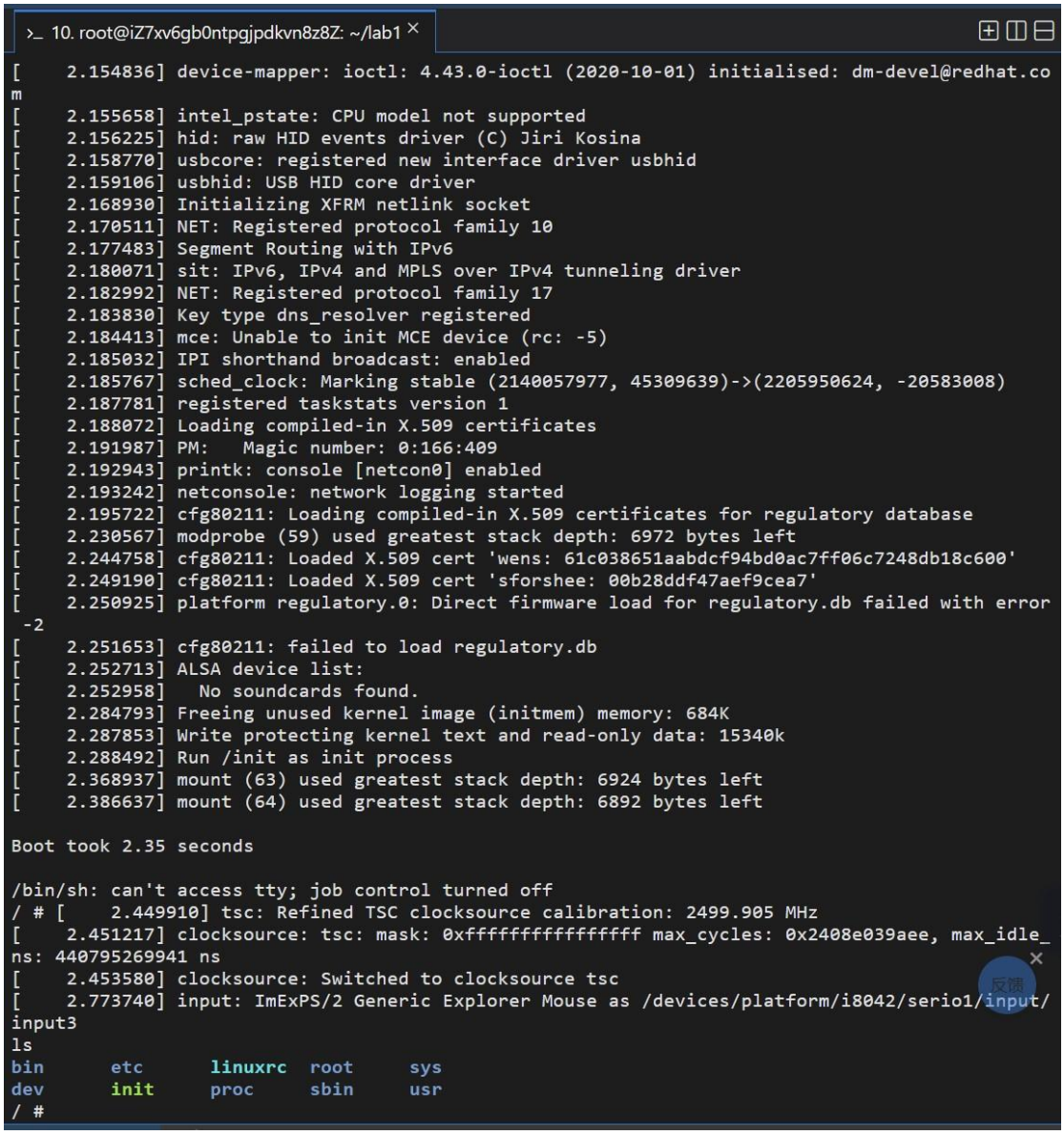
```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
exec /bin/sh
```

让我们逐行解释一下：

1. `#!/bin/sh`：这是脚本的 shebang 行，指定了脚本使用的 Shell 解释器（在这里是 `/bin/sh`）。
2. `mount -t proc none /proc`：这行挂载了 `proc` 文件系统到 `/proc` 目录。`/proc` 是一个虚拟文件系统，提供了有关系统和进程的信息。
3. `mount -t sysfs none /sys`：这行挂载了 `sysfs` 文件系统到 `/sys` 目录。`/sys` 同样是一个虚拟文件系统，用于访问内核和设备的信息。
4. `echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime)`

```
chmod u+x init
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ~/lab1/initramfs-busybox-x86.cpio.gz
cd ~/lab1
qemu-system-i386 -kernel linux-5.10.212/arch/x86/boot/bzImage -initrd initramfs-busybox-x86.cpio.gz - nographic -append "console=ttyS0"
ls
```

结果展示：启动花了 2.35 秒



。

Section 5 实验总结与心得体会

棘手的问题：linux 内核版本不匹配，需要更新比较新的内核版本才可以正常进行 qemu 和 gdb 调试；

心得体会：初步认识到 linux 系统的启动过程，内核（kernel）初始化结束后，initramfs 作为临时根文件系统被挂载，helloworld 是最简单的一类，还没有实现根文件的系统，mybusybox 中基于已有程序给出了根文件系统，使得操作系统可以正常启动。

Section 6 对实验的改进建议和意见

1，实验中的推荐使用 linux-3 的版本是否和最新的 gdb 和 qemu 平台不兼容？

2，实验中的原理还需要阐释的更明白一些，很多命令行原理和 Linux 系统启动知识需要上网搜索，对于一些同学可能是不友好的。

Section 7 附录：参考资料清单

本节为可选章节，可以列出自己在实验过程中的一些重要参考书籍、博客网站等等，为将来的实验提供帮助。

linux 启动流程——initrd 和 initramfs_initrd-switch-root" "initramfs-CSDN 博客

Section 8 附录：代码清单

【实验任务 3】完整的 c 程序如下：

```
Vim helloworld.c  #使用 vim 编辑代码
#include void main()
{     printf("lab1: Hello World\n");
      fflush(stdout); /* 让程序打印完后继续维持在用户态 */
      while(1);
 }
```