本科生实验报告

实验课程:_____操作系统原理实验_____

实验名称:___实验 1 编译内核/利用已有内核构建 OS_____

专业名称:_____计算机科学与技术(超级计算方向) _____

学生姓名:_____林天皓_____

学生学号:_____18324034_____

实验地点:_____

实验成绩:_____

报告时间:_____2021.3.4_____

操作系统 实验1 编译内核/利用已有内核构建OS

林天皓 18324034

实验要求

编译内核利用已有内核构建OS 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；

需求内容

1. 搭建OS内核开发环境包括：代码编辑环境、编译环境、运行环境、调试环境等；
2. 下载并编译i386（32位）内核，并利用qemu启动内核；
3. 熟悉制作initramfs的方法；
4. 编写简单应用程序随内核启动运行；
5. 编译i386版本的Busybox，随内核启动，构建简单的OS；
6. 开启远程调试功能，进行调试跟踪代码运行；
7. 撰写实验报告；

实验步骤与结果

基础环境搭建

本人实验过程中使用的环境为，由于以前已配置好，配置过程略

虚拟机vmware workstation pro 16.1

操作系统 Ubuntu 18.04.5 LTS

C语言环境 gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0 gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1

rust环境 rustc 1.50.0 (cb75ad5db 2021-02-10)

编辑环境 使用宿主机vscode插件remote development远程编辑

qemu环境 QEMU emulator version 5.0.0（源码安装）

> - 编译安装qemu tips
> - 出现 `ERROR: pkg-config binary 'pkg-config' not found` 时，可以通过 `sudo apt-get install pkg-config` 安装；
> - 出现 `ERROR: glib-2.48 gthread-2.0 is required to compile QEMU` 时，可以通过 `sudo apt-get install libglib2.0-dev` 安装；
> - 出现 `ERROR: pixman >= 0.21.8 not present` 时，可以通过 `sudo apt-get install libpixman-1-dev` 安装。

> - busybox安装tips
> - 出现错误 curses.h：No such file or directory ， 通过sudo apt-get install libncurses5-dev libncursesw5-dev安装

下载编译linux kernel并qemu启动内核

编译

```
make i386_defconfig #载入默认x86配置
make menuconfig #修改debug属性，可以被调试
make -j$(nproc) #动态指定最大的cpu数量
```

执行完毕后生成符号表linux-5.10.19/vmlinux

与核心文件 linux-5.10.19/arch/x86/boot/bzImage

qemu执行

```
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -S -append
"console=ttyS0" -nographic
```

执行后无任何输出，但通过ps -a|grep qemu可以看到已经启动执行

```
lthos@ubuntu:~/os/exp1/hello_rs$ ps -a |grep qemu
   5577 pts/3    00:00:00 qemu-system-i38
```

gdb调试

gdb执行命令

```
file vmlinux #载入符号表
break start_kernel #在start_kernel设置断点
target remote:1234 #连接程序端口
c #执行程序直到断点
```

```
lthos@ubuntu:~/os/exp1/linux-5.10.19$ gdb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file vmlinux
Reading symbols from vmlinux...(no debugging symbols found)...done.
(gdb) break start_kernel
Breakpoint 1 at 0xc1fb0815
(gdb) target remote:1234
Remote debugging using :1234
0x0000fff0 in ?? ()
(gdb) c
Continuing.
```

执行之后qemu有linux的启动输出

节选后一部分：

```
[    3.312754] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[    3.315306] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.10.19 #1
[    3.316493] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.13.0-0-gf21b5a4aeb02-prebuilt.qemu.o
[    3.317773] Call Trace:
[    3.319084]  dump_stack+0x54/0x68
[    3.319565]  panic+0x9e/0x251
[    3.319916]  mount_block_root+0x133/0x1b3
[    3.320395]  mount_root+0xd3/0xec
[    3.320747]  prepare_namespace+0x116/0x141
[    3.321217]  kernel_init_freeable+0x19c/0x1a9
[    3.321683]  ? rest_init+0xa0/0xa0
[    3.322080]  kernel_init+0x8/0xf0
[    3.322453]  ret_from_fork+0x1c/0x28
[    3.323863] Kernel Offset: disabled
[    3.324771] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

可见因为没有挂载根目录导致内核panic未能启动，下一步我们加入initramfs载入

制作initramfs

在尝试编写我们的内核启动的程序的时候，我们尝试使用c语言和rust语言编译成为32位可执行文件。

首先我们使用C语言建立我们的initramfs。

C语言hello_c

编辑一个C语言文件

hello_c > C hello.c

```c
#include<stdio.h>
int main(){
    printf("hello from c language\n");
    return 0;
}

```

使用gcc的`-m32`选项将程序编译为32位

```
lthos@ubuntu:~/os/exp1/hello_c$ gcc -o helloworld -m32 -static hello.c
```

通过readelf命令查看编译后程序信息

```
lthos@ubuntu:~/os/exp1/hello_c$ readelf -h helloworld
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048730
  Start of program headers:          52 (bytes into file)
  Start of section headers:          656316 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
  Size of section headers:           40 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

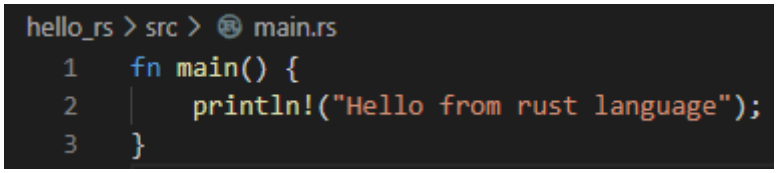
可见编译结果程序为ELF32的可执行文件，机器为intel 80386

rust语言hello_rs

由于需要交叉编译，需要通过rustup引入新目标架构

```
cargo new hello_rs #新建rust项目
rustup target add i686-unknown-linux-musl#添加目标架构为i686静态
```

修改main.rs



需要设置--target参数使得rust编译为静态目标

```
cargo build --target i686-unknow-linux-musl #编译为i686架构静态目标
readelf -h ./target/i686-unknown-linux-musl/debug/hello_rs #读取elf文件头部信息
```

```
llo_rsubuntu:~/os/exp1/hello_rs$ readelf -h target/i686-unknown-linux-musl/debug/h
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80481aa
  Start of program headers:          52 (bytes into file)
  Start of section headers:          3763608 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         29
  Section header string table index: 28
```

可见和c语言编译的文件一样都是ELF32文件。

接下来我们尝试通过initramfs分别载入我们刚才编译的这两个文件

```
lthos@ubuntu:~/os/exp1/hello_c$ echo helloworld | cpio -o --format=newc > hwinitramfs
1285 blocks
```

```
lthos@ubuntu:~/os/exp1/hello_rs$ echo hello_rs | cpio -o --format=newc > hwinitramfs_rs
7354 blocks
```

载入qemu执行

```
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd
hello_c/hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -nographic
```

接下来通过另一个终端以前文的方式gdb调试内核，此处不再赘述

```
[    3.536146] PM:   Magic number: 13:463:975
[    3.537856] printk: console [netcon0] enabled
[    3.538474] netconsole: network logging started
[    3.545303] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[    3.576284] kworker/u2:0 (59) used greatest stack depth: 6996 bytes left
[    3.605917] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    3.609654] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[    3.612940] ALSA device list:
[    3.615402]   No soundcards found.
[    3.618203] cfg80211: failed to load regulatory.db
[    3.798167] Freeing unused kernel image (initmem) memory: 672K
[    3.800780] Write protecting kernel text and read-only data: 14052k
[    3.805063] Run helloworld as init process
hello from c language
[    3.855886] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[    3.857504] CPU: 0 PID: 1 Comm: helloworld Not tainted 5.10.19 #1
[    3.858574] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.13.0-0-gf21b5a4aeb02-prebuilt.qemu.org 04/01/2014
[    3.860143] Call Trace:
[    3.905828]  dump_stack+0x54/0x68
[    3.906248]  panic+0x9e/0x251
[    3.906765]  do_exit+0x724/0xa80
[    3.906926]  ? tick_program_event+0x39/0x90
[    3.907413]  ? hrtimer_interrupt+0x127/0x280
[    3.908276]  do_group_exit+0x30/0x90
[    3.909033]   ia32_sys_exit_group+0x10/0x10
```

成功输出hello from c language,与预期现象相同。

rust语言版本相同，此处仅展示结果

```
[    3.338978] cfg80211: failed to load regulatory.db
[    3.340831] ALSA device list:
[    3.341464]   No soundcards found.
[    3.513857] Freeing unused kernel image (initmem) memory: 672K
[    3.533696] Write protecting kernel text and read-only data: 14052k
[    3.547352] Run hello_rs as init process
Hello from rust language
[    3.595915] Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000000
[    3.597571] CPU: 0 PID: 1 Comm: hello_rs Not tainted 5.10.19 #1
[    3.598071] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.13.0-0-gf21b5a4aeb02-pre
[    3.600212] Call Trace:
[    3.601855]  dump_stack+0x54/0x68
[    3.602441]  panic+0x9e/0x251
[    3.602718]  do_exit+0x724/0xa80
```

编译并启动Busybox

与上文中的程序一致，busybox需要编译为32位i386架构静态可执行文件。

```
make defconfig #载入默认设置
make menuconfig
```

通过设置编译为静态文件，配置CFLAGS为-m32 和LDFLAGS为-m32 -march=i386。

```
make -j$(nproc)
make install
```

编译并安装，接下来制作busybox的Initramfs

```
cd ~/lab1
mkdir mybusybox
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
cp -av busybox-1_33_0/_install/* mybusybox/
cd mybusybox
```

创建mybusybox文件夹并拷贝busybox-1_33_0/_install/中的所有文件

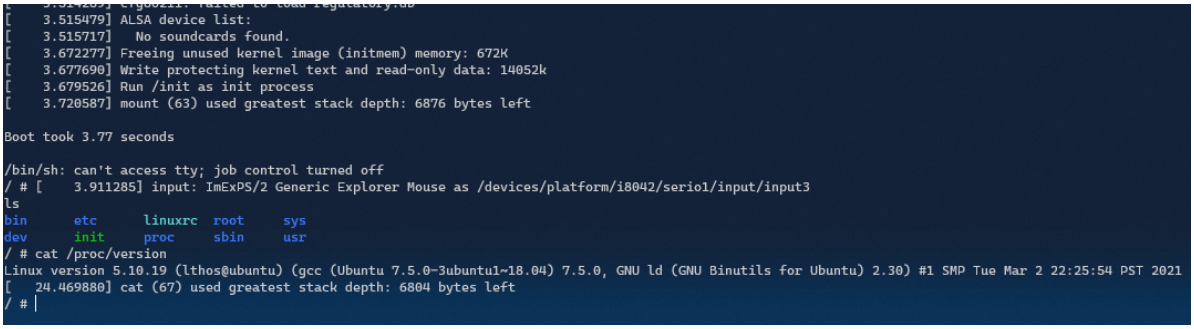
然后创建init文件作文linux启动后执行的命令

```
#!/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
```

```
chmod u+x init #修改init文件所有者的执行权限
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ~/lab1/initramfs-busybox-x86.cpio.gz #创建Initramfs
```

```
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0" #qemu启动
```

经过一段时间的启动，成功执行了我们设置的init文件



成功进入busybox的/bin/sh,经过测试简单的ls和cat指令运行正常

总结

本次实验中，了解了基本linux命令的执行，学习了如何在linux环境中从源码手动配置，编译与安装软件。尝试了用C语言和Rust语言分别编译为32位静态可执行文件并载入linux内核在模拟器中成功执行。得益于实验指导的详细步骤，在整个实验过程中并没有遇到很多的问题。在下一步中，由于本人对Rsut语言并不熟悉，我打算以C语言为基础，c语言实现完成之后再翻译为rust。并了解与对比x86,arm与riscv在一些功能中的差异。

参考资料

https://qemu-project.gitlab.io/qemu/devel/build-system.html

https://www.cnblogs.com/wipan/p/9264979.html

https://doc.rust-lang.org/cargo/reference/specifying-dependencies.html?highlight=64#platform-specific-dependencies

https://www.cnblogs.com/dhcn/p/12123205.html

https://blog.csdn.net/castellan/article/details/86063775