



本科生实验报告

实验课程: 操作系统实验

实验名称: lab1 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

学生姓名: 杜翊菲

学生学号: 23336062

实验地点: 实验大楼 B201

实验成绩:

报告时间: 2025 年 3 月 5 日

Lab1 编译内核/利用已有内核构建 OS

一、实验要求

1. 独立完成实验 5 个部分：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. 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
3. 下载并编译 i386 (32 位) 内核，并利用 qemu 启动内核
4. 熟悉制作 initramfs 的方法
5. 编写简单应用程序随内核启动运行
6. 编译 i386 版本的 Busybox，随内核启动，构建简单的 OS
7. 开启远程调试功能，进行调试跟踪代码运行
8. 撰写实验报告

二、实验过程

分为五个部分，每个部分包括实验操作、关键代码和结果截图。

(一) 环境配置

1. 安装 Linux 环境

注意：本教程的所有实验均在ubuntu环境进行 (linux的常见版本)。由于大家平时使用的多是 windows 系统，推荐下载virtualbox虚拟机 (vmware, wsl也可行)，并在虚拟机中安装ubuntu 系统进行实验(参考教程)。也可以安装双系统。

在非Linux环境下，首先下载安装Virtualbox，用于启动虚拟机。建议安装Ubuntu 18.04桌面版。

virtualbox: <https://download.virtualbox.org/virtualbox/7.0.6/VirtualBox-7.0.6-155176-Win.exe>

ubuntu18.04: <https://mirrors.tuna.tsinghua.edu.cn/ubuntu-releases/18.04.6/ubuntu-18.04.6-desktop-amd64.iso>

已按要求下载安装 Virtualbox 和 Ubuntu18.04

2. 还源

确保安装好ubuntu环境。进入ubuntu环境后，开始下面的实验。

由于ubuntu的下载源默认是国外的，为了提高下载速度，我们需要将下载源更换为国内源。我们首先备份原先的下载源。

```
sudo mv /etc/apt/sources.list /etc/apt/sources.list.backup
```

然后找到清华的ubuntu下载源。注意，选择你ubuntu的版本对应的下载源。



清华大学开源软件镜像站

HOMEEVENTSBLOGRSSPODCASTMIRRORS

AOSP
AUR
AdoptOpenJDK
CPAN
CRAN
CTAN
CocoaPods
alpine
anaconda
anthon
arch4edu
archlinux

Ubuntu 镜像使用帮助

Ubuntu 的软件源配置文件是 `/etc/apt/sources.list`。将系统自带的该文件做个备份，将该文件替换为下面内容，即可使用 TUNA 的软件源镜像。

选择你的ubuntu版本:

18.04 LTS

```
# 默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
```

本镜像仅包含 32/64 位 x86 架构处理器的软件包。在 ARM(arm64, armhf)、PowerPC(ppc64el)、RISC-V(riscv64) 和 S390x 等架构的设备上（对应官方源为ports.ubuntu.com）请使用 [ubuntu-ports 镜像](#)。

然后使用 `gedit` 打开下载源保存的文件 `/etc/apt/sources.list`

```
sudo gedit /etc/apt/sources.list
```

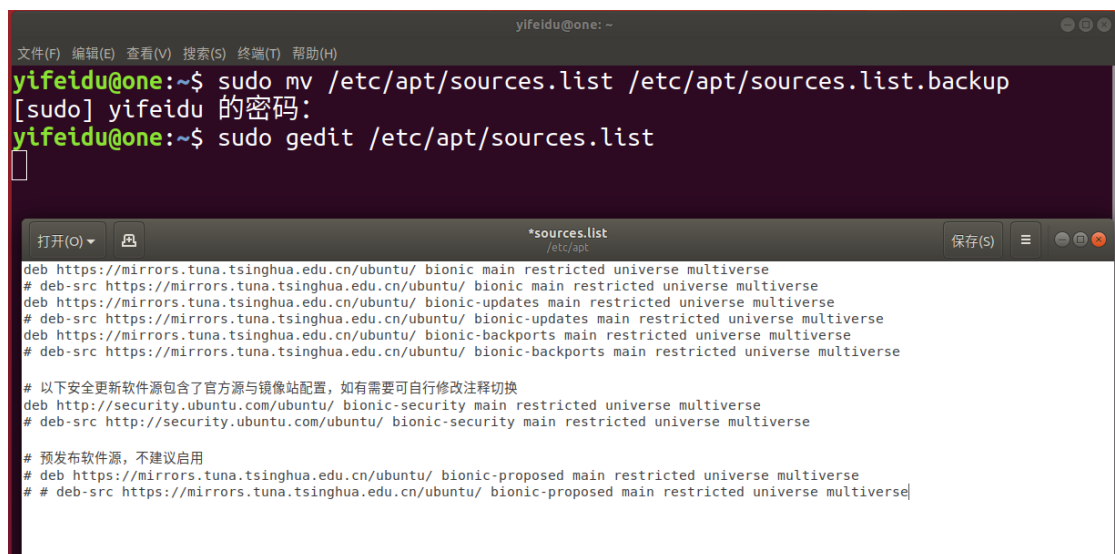
将下载源复制进 `/etc/apt/sources.list` 后保存退出。

更新 `apt`，检查是否更换为清华的下载源。

```
sudo apt update
```

替换下载源过程截图如下：

(1) 下载源复制进`/etc/apt/sources.list`后保存退出



(2) 更新 `apt` 后如图所示

```
yifeidu@one: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
d64 Packages [18.2 kB]  
获取:63 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/universe i3  
86 Packages [18.1 kB]  
获取:64 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/universe Tr  
anslation-en [8,668 B]  
获取:65 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/universe am  
d64 DEP-11 Metadata [10.0 kB]  
获取:66 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/universe DE  
P-11 48x48 Icons [5,009 B]  
获取:67 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/universe DE  
P-11 64x64 Icons [6,808 B]  
获取:68 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/multiverse  
amd64 DEP-11 Metadata [216 B]  
获取:69 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/multiverse  
DEP-11 48x48 Icons [29 B]  
获取:70 https://mirrors.tuna.tsinghua.edu.cn/ubuntu bionic-backports/multiverse  
DEP-11 64x64 Icons [29 B]  
命中:71 http://security.ubuntu.com/ubuntu bionic-security InRelease  
已下载 53.4 MB, 耗时 33秒 (1,640 kB/s)  
正在读取软件包列表... 完成  
正在分析软件包的依赖关系树  
正在读取状态信息... 完成  
有 311 个软件包可以升级。请执行 'apt list --upgradable' 来查看它们。  
yifeidu@one:~$
```

3. 配置 C/C++ 环境

```
sudo apt install binutils  
sudo apt install gcc
```

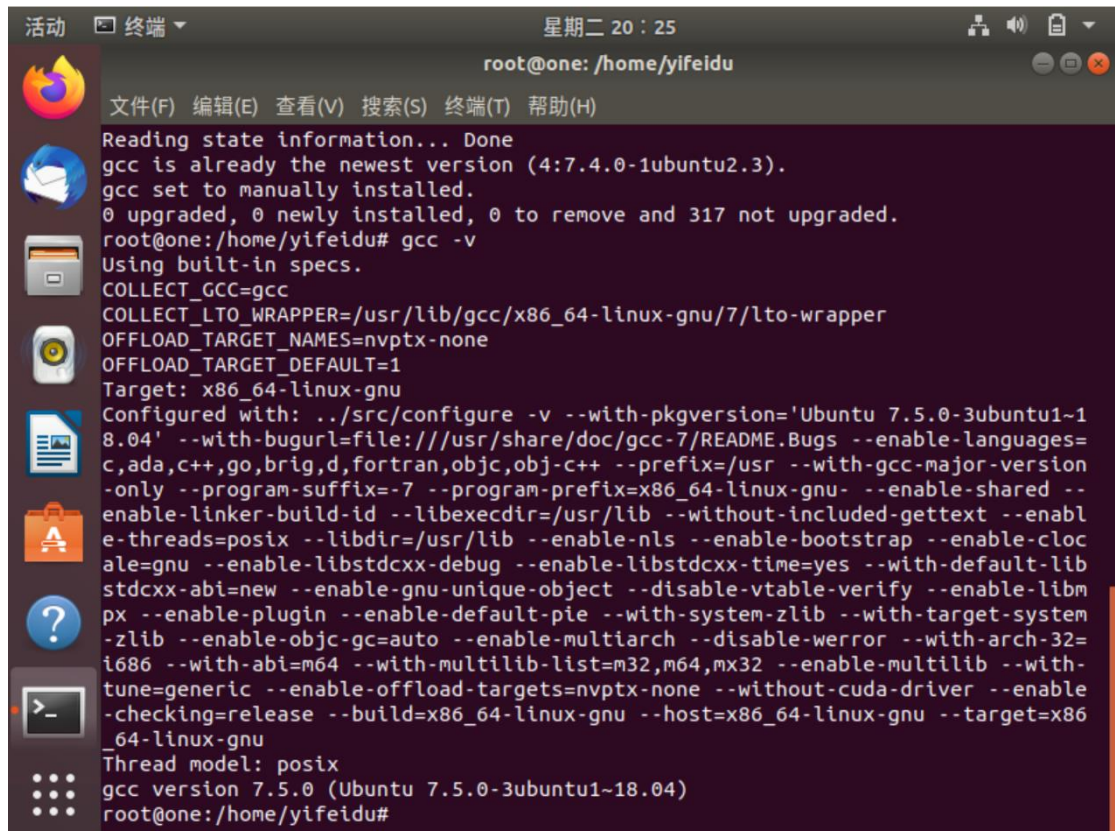
查看 gcc 是否安装。

```
gcc -v
```

若输出gcc的版本号则表明安装成功。

过程截图如下：

```
root@one:/home/yifeidu# visudo  
root@one:/home/yifeidu# sudo apt install binutils  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  binutils-common binutils-x86-64-linux-gnu libbinutils  
Suggested packages:  
  binutils-doc  
The following packages will be upgraded:  
  binutils binutils-common binutils-x86-64-linux-gnu libbinutils  
4 upgraded, 0 newly installed, 0 to remove and 317 not upgraded.  
Need to get 2,529 kB of archives.  
After this operation, 0 B of additional disk space will be used.  
Do you want to continue? [Y/n]
```



```
活动 终端 星期二 20:25
root@one: /home/yifeidu
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Reading state information... Done
gcc is already the newest version (4:7.4.0-1ubuntu2.3).
gcc set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 317 not upgraded.
root@one:/home/yifeidu# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-cloCALE=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmplex --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
root@one:/home/yifeidu#
```

4. 安装其他工具

```
sudo apt install nasm
sudo apt install qemu
sudo apt install cmake
sudo apt install libncurses5-dev
sudo apt install bison
sudo apt install flex
sudo apt install libssl-dev
sudo apt install libc6-dev-i386
```

安装vscode以及在vscode中安装汇编、C/C++插件。vscode将作为一个有力的代码编辑器。
已自行安装完毕

(二) 编译 Linux 内核

1. 下载内核

我们先在当前用户目录下创建文件夹 lab1 并进入。或者直接使用实验仓库的 lab1 文件夹。

```
mkdir ~/lab1
cd ~/lab1
```


到 <https://www.kernel.org/> 下载内核5.10.x到文件夹 `~/lab1`。

(这里我们提供一种无需进入网页, 快捷的方式下载内核, 可在终端下输入如下指令)

```
wget https://mirrors.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.19.tar.xz
```

解压并进入。

```
xz -d linux-5.10.19.tar.xz
tar -xvf linux-5.10.19.tar
cd linux-5.10.19
```

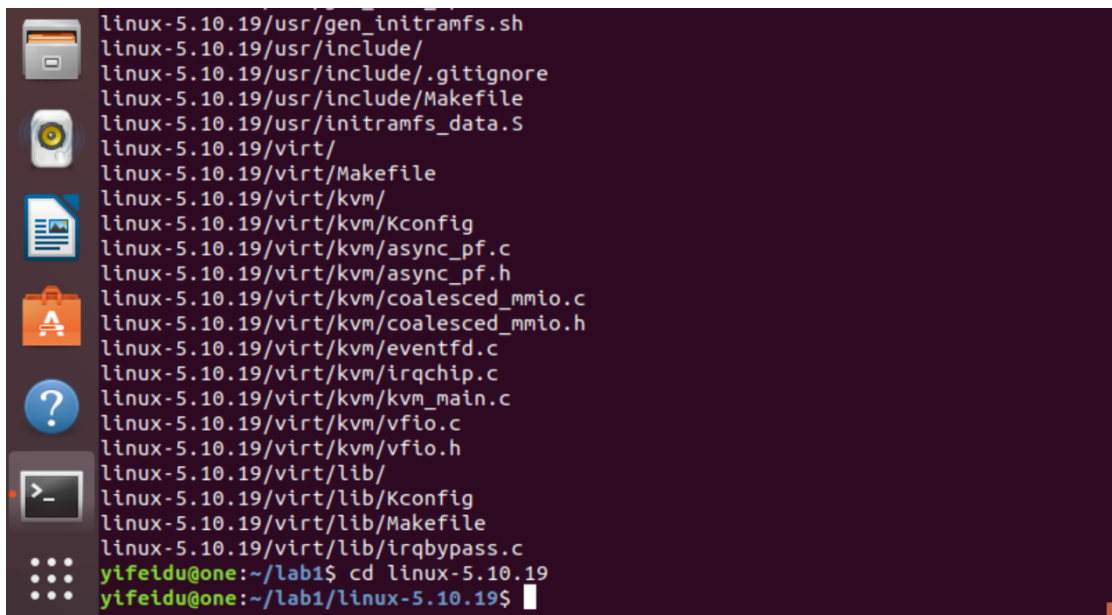
下载并解压过程截图如下:

(1) 下载内核:



```
yifeidu@one:~$ mkdir ~/lab1
yifeidu@one:~$ cd ~/lab1
yifeidu@one:~/lab1$ wget https://mirrors.yuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.10.tar.xz
--2025-03-04 20:45:26-- https://mirrors.yuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.10.tar.xz
正在解析主机 mirrors.yuna.tsinghua.edu.cn (mirrors.yuna.tsinghua.edu.cn)... 失败: 未知的名称或服务。
wget: 无法解析主机地址 "mirrors.yuna.tsinghua.edu.cn"
yifeidu@one:~/lab1$ wget https://mirrors.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.19.tar.xz
--2025-03-04 20:46:23-- https://mirrors.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.19.tar.xz
正在解析主机 mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)... 101.6.15.130, 2402:f000:1:400::2
正在连接 mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)|101.6.15.130|:443... 已连接。
已发出 HTTP 请求, 正在等待回应... 200 OK
长度: 116276000 (111M) [application/octet-stream]
正在保存至: "linux-5.10.19.tar.xz"
linux-5.10.19.tar.x 52%[=====> ] 58.68M 4.12MB/s 剩余 14s
```

(2) 解压完成:



```
linux-5.10.19/usr/gen_initramfs.sh
linux-5.10.19/usr/include/
linux-5.10.19/usr/include/.gitignore
linux-5.10.19/usr/include/Makefile
linux-5.10.19/usr/initramfs_data.S
linux-5.10.19/virt/
linux-5.10.19/virt/Makefile
linux-5.10.19/virt/kvm/
linux-5.10.19/virt/kvm/Kconfig
linux-5.10.19/virt/kvm/async_pf.c
linux-5.10.19/virt/kvm/async_pf.h
linux-5.10.19/virt/kvm/coalesced_mmio.c
linux-5.10.19/virt/kvm/coalesced_mmio.h
linux-5.10.19/virt/kvm/eventfd.c
linux-5.10.19/virt/kvm/irqchip.c
linux-5.10.19/virt/kvm/kvm_main.c
linux-5.10.19/virt/kvm/vfio.c
linux-5.10.19/virt/kvm/vfio.h
linux-5.10.19/virt/lib/
linux-5.10.19/virt/lib/Kconfig
linux-5.10.19/virt/lib/Makefile
linux-5.10.19/virt/lib/irqbypass.c
yifeidu@one:~/lab1$ cd linux-5.10.19
yifeidu@one:~/lab1/linux-5.10.19$
```

2. 编译内核

将内核编译成i386 32位版本。

```
make i386_defconfig
make menuconfig
```

在打开的图像界面中依次选择 Kernel hacking 、 Compile-time checks and compiler options , 最后在 [] Compile the kernel with debug info 输入 Y 勾选, 保存退出。

编译内核, 这一步较慢。

```
make -j8
```

检查Linux压缩镜像 linux-5.10.19/arch/x86/boot/bzImage 和符号表 linux-5.10.19/vmlinux 是否已经生成。

过程截图如下:

(1) 将内核编译成 i386 32 位版本

```
yifeidu@one:~/lab1/linux-5.10.19$ make i386_defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdata.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
#
# configuration written to .config
#
yifeidu@one:~/lab1/linux-5.10.19$
```

(2) 编译内核

```
CC arch/x86/boot/compressed/acpi.o
HOSTCC arch/x86/boot/tools/build
CPUSTR arch/x86/boot/cpustr.h
CC arch/x86/boot/cpu.o
CC arch/x86/boot/compressed/misc.o
GZIP arch/x86/boot/compressed/vmlinux.bin.gz
MKPIGGY arch/x86/boot/compressed/piggy.S
AS arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS arch/x86/boot/header.o
LD arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#1)
yifeidu@one:~/lab1/linux-5.10.19$
```

(三) 启动内核并调试

1. 启动 qemu

使用 `qemu` 启动内核并开启远程调试。

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

此时，同学们会发现qemu并未输出任何信息。这是因为我们开启了gdb调试，而qemu在等待gdb输入的指令后才能继续执行。接下来我们启动gdb，通过gdb来告诉qemu应该怎么做。

此时没有任何输出如下图：

```
yifeldu@one:~/lab1/linux-5.10.19$ cd ~/lab1
yifeldu@one:~/lab1$ qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic
qemu-system-i386-kernel: 未找到命令
yifeldu@one:~/lab1$ qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic
qemu-system-i386: -5.10.19/arch/x86/boot/bzImage: invalid option
yifeldu@one:~/lab1$
```

2. gdb 调试

在另外一个Terminal下启动gdb，注意，不要关闭qemu所在的Terminal。

```
cd ~/lab1
gdb
```

在gdb下，加载符号表

```
file linux-5.10.19/vmlinux
```

在gdb下，连接已经启动的qemu进行调试。

```
target remote:1234
```

在gdb下，为start_kernel函数设置断点。

```
break start_kernel
```

在gdb下，输入 `c` 运行。

```
c
```

调试截图如下：

(1) 启动 gdb 的新终端页面:

```
yifeidu@one: ~/lab1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
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 linux-5.10.19/vmlinux
Reading symbols from linux-5.10.19/vmlinux...done.
(gdb) target remote:1234
Remote debugging using :1234
0x0000fff0 in ?? ()
(gdb) break start_kernel
Breakpoint 1 at 0xc1fac80f: file init/main.c, line 849.
(gdb) c
Continuing.
```

(2) 原终端页面:

```
yifeidu@one: ~/lab1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
r -6
[ 3.180954] Please append a correct "root=" boot option; here are the available partitions:
[ 3.181906] 0b00          1048575 sr0
[ 3.181954] driver: sr
[ 3.182433] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[ 3.183021] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.10.19 #1
[ 3.183637] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
[ 3.184181] Call Trace:
[ 3.185333] dump_stack+0x54/0x68
[ 3.185534] panic+0x9e/0x247
[ 3.185669] mount_block_root+0x133/0x1b3
[ 3.185840] mount_root+0xd3/0xec
[ 3.186119] prepare_namespace+0x116/0x141
[ 3.186438] kernel_init_freeable+0x19c/0x1a9
[ 3.186640] ? rest_init+0xa0/0xa0
[ 3.186889] kernel_init+0x8/0xf0
[ 3.187052] ret_from_fork+0x1c/0x28
[ 3.187815] Kernel Offset: disabled
[ 3.188264] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

(四) 制作 Initramfs

1. Hello World

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，

```
vim helloworld.c
```

Hello World程序如下。

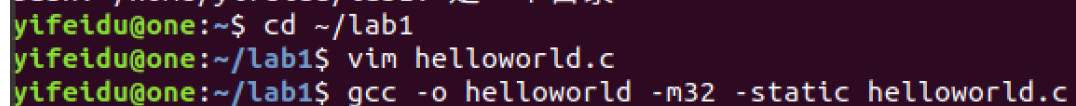
```
#include <stdio.h>

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);
}
```

上述文件保存在 ~/lab1/helloworld.c 中，然后将上面代码编译成32位可执行文件。

```
gcc -o helloworld -m32 -static helloworld.c
```

运行截图如下：



```
yifeidu@one:~$ cd ~/lab1
yifeidu@one:~/lab1$ vim helloworld.c
yifeidu@one:~/lab1$ gcc -o helloworld -m32 -static helloworld.c
```

2. 加载 Initramfs

用cpio打包initramfs。

```
echo helloworld | cpio -o --format=newc > hwinitramfs
```

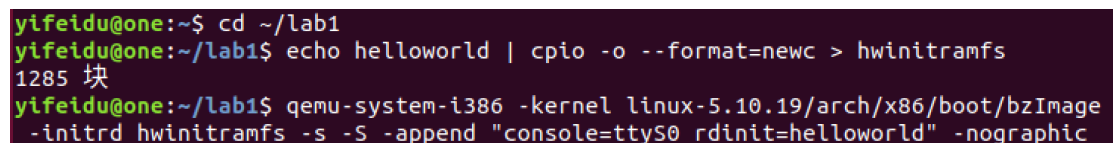
启动内核，并加载initramfs。

```
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd hwinitramfs -s
```

重复上面的gdb的调试过程，可以看到qemu中输出了 lab1: Hello World\n

该过程截图如下：

(1) 启动 gdb 前



```
yifeidu@one:~$ cd ~/lab1
yifeidu@one:~/lab1$ echo helloworld | cpio -o --format=newc > hwinitramfs
1285 块
yifeidu@one:~/lab1$ qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage
-initrd hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -nographic
```

(2) 启动 gdb 后的输出:

```
yifeidu@one: ~/lab1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[ 1.693078] PM: Magic number: 1:124:670
[ 1.693252] workqueue scsi_tmf_0: hash matches
[ 1.693947] printk: console [netcon0] enabled
[ 1.694190] netconsole: network logging started
[ 1.697280] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[ 1.706329] kworker/u2:0 (59) used greatest stack depth: 7156 bytes left
[ 1.716407] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[ 1.718974] platform regulatory.0: Direct firmware load for regulatory.db failed with error -2
[ 1.719269] cfg80211: failed to load regulatory.db
[ 1.720287] ALSA device list:
[ 1.720737] No soundcards found.
[ 1.749969] Freeing unused kernel image (initmem) memory: 672K
[ 1.753743] Write protecting kernel text and read-only data: 14036k
[ 1.754047] Run helloworld as init process
lab1: Hello World
[ 1.915465] tsc: Refined TSC clocksource calibration: 2993.653 MHz
[ 1.916068] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x2b26d9d3eac, max_idle_ns: 440795332698 ns
[ 1.916455] clocksource: Switched to clocksource tsc
[ 2.281141] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3
```

(五) 编译并启动 Busybox

1. 下载并解压

解压 `~/lab1` 中的Busybox。

```
tar -xf Busybox_1_33_0.tar.gz
```

2. 编译 busybox

```
cd busybox-1_33_0
make defconfig
make menuconfig
```

进入 `settings` , 然后在 `Build static binary(no shared libs)` 处输入 `Y` 勾选, 然后 `ENTER` 选中 `() Additional CFLAGS` 并键入 `-m32 -march=i386` ,

同理在 `() Additional LDFLAGS` 键入 `-m32` 。

保存退出, 然后编译。

```
make -j8
make install
```

操作过程截图如下：

```
yifeidu@one: ~/lab1/busybox-1_33_0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
CC      libbb/wfopen_input.o
CC      libbb/write.o
CC      libbb/xatoum.o
CC      libbb/xconnect.o
CC      libbb/xfunc_die.o
CC      libbb/xfuncs.o
CC      libbb/xfuncs_printf.o
CC      libbb/xgetcwd.o
CC      libbb/xgethostbyname.o
CC      libbb/xreadlink.o
CC      libbb/xrealloc_vector.o
CC      libbb/xregcomp.o
AR      libbb/lib.a
LINK    busybox_unstripped
Static linking against glibc, can't use --gc-sections
Trying libraries: crypt m resolv rt
Library crypt is not needed, excluding it
Library m is needed, can't exclude it (yet)
Library resolv is needed, can't exclude it (yet)
Library rt is not needed, excluding it
Library m is needed, can't exclude it (yet)
Library resolv is needed, can't exclude it (yet)
Final link with: m resolv
yifeidu@one:~/lab1/busybox-1_33_0$
```

```
yifeidu@one: ~/lab1/busybox-1_33_0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
./_install//usr/sbin/rpcwakeup -> ../../bin/busybox
./_install//usr/sbin/sendmail -> ../../bin/busybox
./_install//usr/sbin/setfont -> ../../bin/busybox
./_install//usr/sbin/setlogcons -> ../../bin/busybox
./_install//usr/sbin/svlogd -> ../../bin/busybox
./_install//usr/sbin/telnetd -> ../../bin/busybox
./_install//usr/sbin/tftpd -> ../../bin/busybox
./_install//usr/sbin/ubiattach -> ../../bin/busybox
./_install//usr/sbin/ubidetach -> ../../bin/busybox
./_install//usr/sbin/ubimkvol -> ../../bin/busybox
./_install//usr/sbin/ubirename -> ../../bin/busybox
./_install//usr/sbin/ubirmvol -> ../../bin/busybox
./_install//usr/sbin/ubirsvol -> ../../bin/busybox
./_install//usr/sbin/ubiupdatevol -> ../../bin/busybox
./_install//usr/sbin/udhcpd -> ../../bin/busybox

-----
You will probably need to make your busybox binary
setuid root to ensure all configured applets will
work properly.
-----
yifeidu@one:~/lab1/busybox-1_33_0$
```

```
yifeidu@one: ~/lab1/busybox-1_33_0
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
BusyBox 1.33.0 Configuration

                                Settings
Arrow keys navigate the menu.  <Enter> selects submenus ---.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

(+)
[ ] Force NOMMU build
[ ] Build shared libbusybox
() Cross compiler prefix
() Path to sysroot
(-m32 -march=i386) Additional CFLAGS
(-m32) Additional LDFLAGS
() Additional LDLIBS
[ ] Avoid using GCC-specific code constructs
[*] Use -mpreferred-stack-boundary=2 on i386 arch
[*] Use -static-libgcc
(+)

<Select>  < Exit >  < Help >
```

3. 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}}
cp -av busybox-1_33_0/_install/* mybusybox/
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

```
vim 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
```

加上执行权限。

```
chmod u+x init
```

将mybusybox中的内容打包归档成cpio文件，以供Linux内核做initramfs启动执行。

```
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ~/lab1/initramfs-busybox
```

操作过程截图如下：



The screenshot shows a terminal window with a dark background. The title bar at the top indicates the file 'init' is open in the directory '~/lab1/mybusybox'. The terminal content shows the script being edited: a shebang line '#!/bin/sh', two 'mount' commands for '/proc' and '/sys', an 'echo' command for boot time, and an 'exec' command to run '/bin/sh'. The text is color-coded: the prompt is blue, 'mount' is red, 'echo' is red, and 'exec' is red.

```
init
~/lab1/mybusybox

#!/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
```



```
yifeidu@one: ~/lab1/mybusybox
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
yifeidu@one:~/lab1/mybusybox$ sudo gedit init
** (gedit:15712): WARNING **: 16:01:52.369: Set document metadata failed
: 不支持设置属性 metadata::gedit-position
yifeidu@one:~/lab1/mybusybox$ chmod u+x init
chmod: 更改'init' 的权限: 不允许的操作
yifeidu@one:~/lab1/mybusybox$ find . -print0 | cpio --null -ov --format=
newc | gzip -9 > ~/lab1/initramfs-busybox-x86.cpio.gz
.
./proc
./usr
./usr/bin
./usr/bin/w
./usr/bin/uniq
./usr/bin/od
./usr/bin/chpst
./usr/bin/diff
./usr/bin/udpsvd
./usr/bin/svok
./usr/bin/seq
./usr/bin/mkpasswd
./usr/bin/setkeycodes
./usr/bin/softlimit
./usr/bin/ts
./usr/bin/whois
./usr/bin/tr
./usr/bin/ttysize
./usr/bin/cryptpw
./usr/bin/tcpshd
./usr/bin/cksum
```

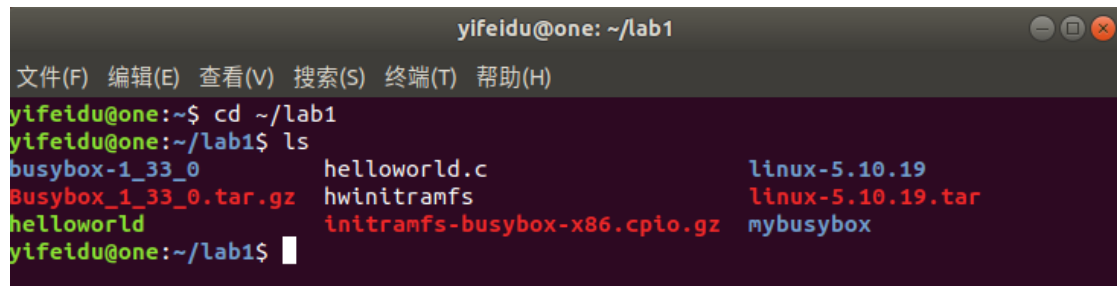
```
yifeidu@one: ~/lab1/mybusybox
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
./sbin/mkswap
./sbin/acpid
./sbin/ifup
./sbin/iproute
./sbin/fdisk
./sbin/raidautorun
./sbin/iprule
./sbin/mkfs.minix
./sbin/blkid
./sbin/ipaddr
./sbin/hdparm
./sbin/iplink
./sbin/mkfs.vfat
./sbin/iptunnel
./sbin/adjtimex
./sbin/syslogd
./sbin/ipneigh
./sbin/ifconfig
./sbin/ifdown
./sbin/modinfo
./sbin/mdev
./sbin/fstrim
./sbin/halt
./sbin/arp
./sbin/mkdosfs
./sbin/uevent
./sbin/rmmod
4304 块
```

4. 加载 Busybox

```
cd ~/lab1  
qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd initramfs-busyb
```

然后使用 `ls` 命令即可看到当前文件夹。

操作后截图如下：



The screenshot shows a terminal window titled 'yifeidu@one: ~/lab1'. The terminal has a menu bar with '文件(F)', '编辑(E)', '查看(V)', '搜索(S)', '终端(T)', and '帮助(H)'. The user has entered the command 'cd ~/lab1' and then 'ls'. The output of 'ls' is displayed in a color-coded format:

busybox-1_33_0	helloworld.c	linux-5.10.19
Busybox_1_33_0.tar.gz	hwinitramfs	linux-5.10.19.tar
helloworld	initramfs-busybox-x86.cpio.gz	mybusybox

The prompt 'yifeidu@one:~/lab1\$' is visible at the bottom.

三、实验总结

问题 1：在 `gdb` 下连接已经启动的 `qemu` 进行调试，但 `target remote:1234` 输入后显示连接超时（解决方法：重启操作系统）

问题 2：`make menuconfig` 失败，没有规则可制作目标（解决方法：先将上一步 `make defconfig` 执行完毕）

诸如此类的问题，归根结底和操作规范和操作细节相关，要求我们仔细输入代码，并检查每一步是否按教程要求完成。遇到特殊情况与教程不相符时，及时求助互联网来检索细节并解决问题。该实验只要按照教程严格执行并及时纠错，基本上能够完整完成。