

操作系统原理实验

实验一 编译内核/利用已有内核构建OS

教师：张青

作业提交

- 加入课堂派在线课程群（和理论课为同一平台）
 - 操作系统原理实验-2024春季学期
- 按照实验要求和实验报告模板完成实验，并撰写实验报告
- 请将实验报告导出为PDF文件，并命名为
 - 学号+姓名.pdf（如21210001李华.pdf）
- DDL之前将**实验报告**和**相关附件（如源码）**提交至作业平台。

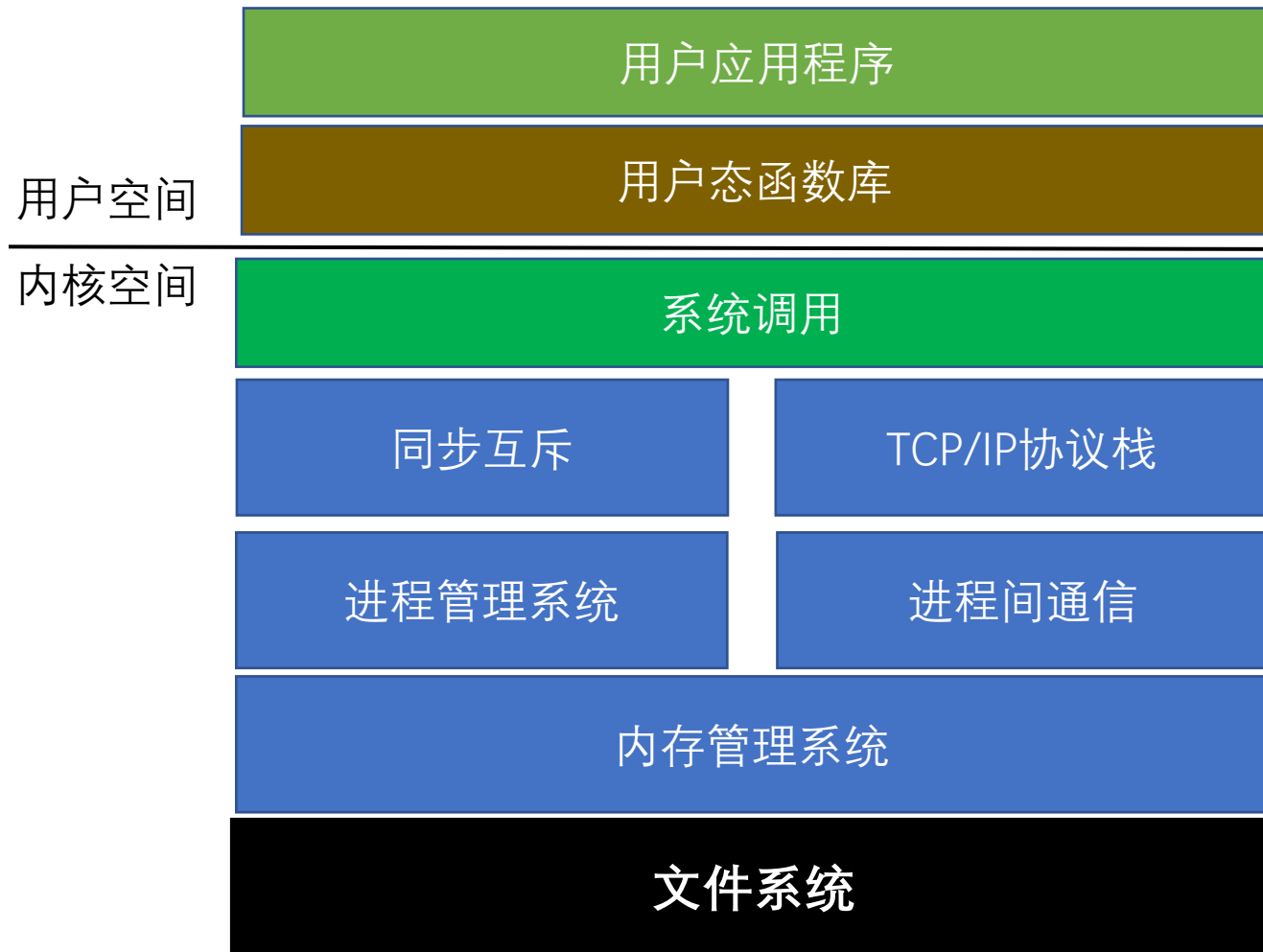
加课码: P5A3DA

微信扫一扫加入课堂



课程实验安排

- 编译内核/利用已有内核构建OS;
- 实模式和保护模式下OS启动;
- OS中断/异常;
- 物理内存管理;
- 虚拟内存管理;
- 内核模式线程管理;
- 用户模式线程管理;
- 系统调用;
- 处理器调度;
- 同步与互斥;
- 文件系统;
- Shell （可选）;
- 迁移到Risc-V架构/或者ARM架构（可选）;



实验一 实验要求

- 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。

实验一 实验概述

- 在本次实验中，同学们会熟悉现有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。同时，同学们会利用精简的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，同学们会熟悉编译环境、相关工具集，并能够实现内核远程调试。
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。

实验一 环境配置

- 搭建Linux系统环境（Ubuntu 18.04）
 - 直接在本机上安装Linux
 - 或者利用VMware或者VirtualBox等
- 更换清华源
 - 备份原本的下载源 `sudo mv /etc/apt/sources.list /etc/apt/sources.list.backup`
 - 找到清华下载源 <https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu>
 - 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（如图代码）复制进/etc/apt/sources.list后保存退出
 - 更新apt `sudo apt update`

手动替换

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 multivers
e
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
```

实验一 环境配置

- 配置C/C++环境

```
sudo apt install binutils
```

```
sudo apt install gcc
```

```
gcc -v
```

- 安装其它工具

```
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
```

实验一 编译Linux内核

- 创建实验一文件夹

```
mkdir ~/lab1
```

```
cd ~/lab1
```

- 到 <https://www.kernel.org/> 下载内核**5.10**到文件夹 ~/lab1

mainline:	6.2	2023-02-19	[tarball]	[pgp]	[patch]	[view diff]	[browse]	
stable:	6.1.13	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.15.95	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.10.169	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	5.4.232	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.19.273	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
longterm:	4.14.306	2023-02-22	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse] [changelog]
linux-next:	next-20230223	2023-02-23						[browse]

- 解压并进入

```
xz -d linux-5.10.169.tar.xz
```

```
tar -xvf linux-5.10.169.tar
```

```
cd linux-5.10.169
```

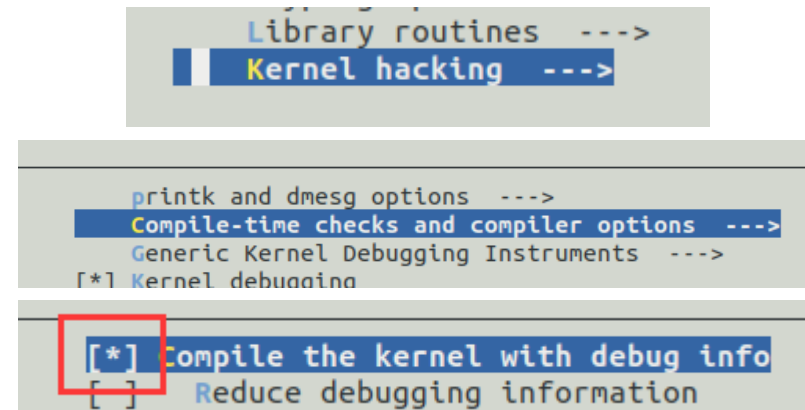

实验一 编译Linux内核

- 编译内核

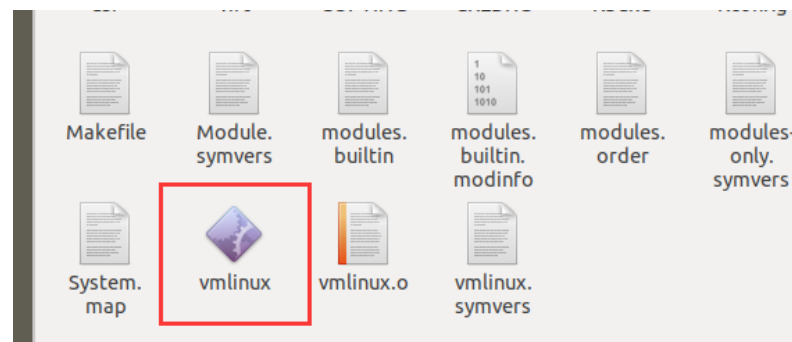
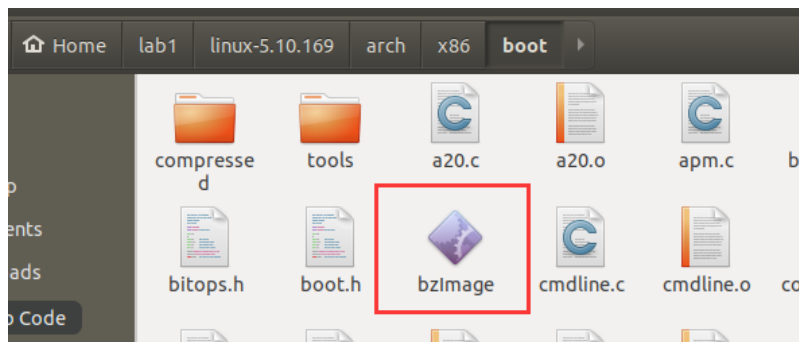
`make i386_defconfig`

`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-5.10.169/arch/x86/boot/bzImage和符号表linux-5.10.169/vmlinux是否已经生成



实验一 启动内核并调试

- 进入文件夹 ~/lab1

`cd ~/lab1`

```
echo@ubuntu:~$ cd ~/lab1
echo@ubuntu:~/lab1$ qemu-system-i386 -kernel linux-5.10.169/arch/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic
```

- 使用qemu启动内核并开启远程调试

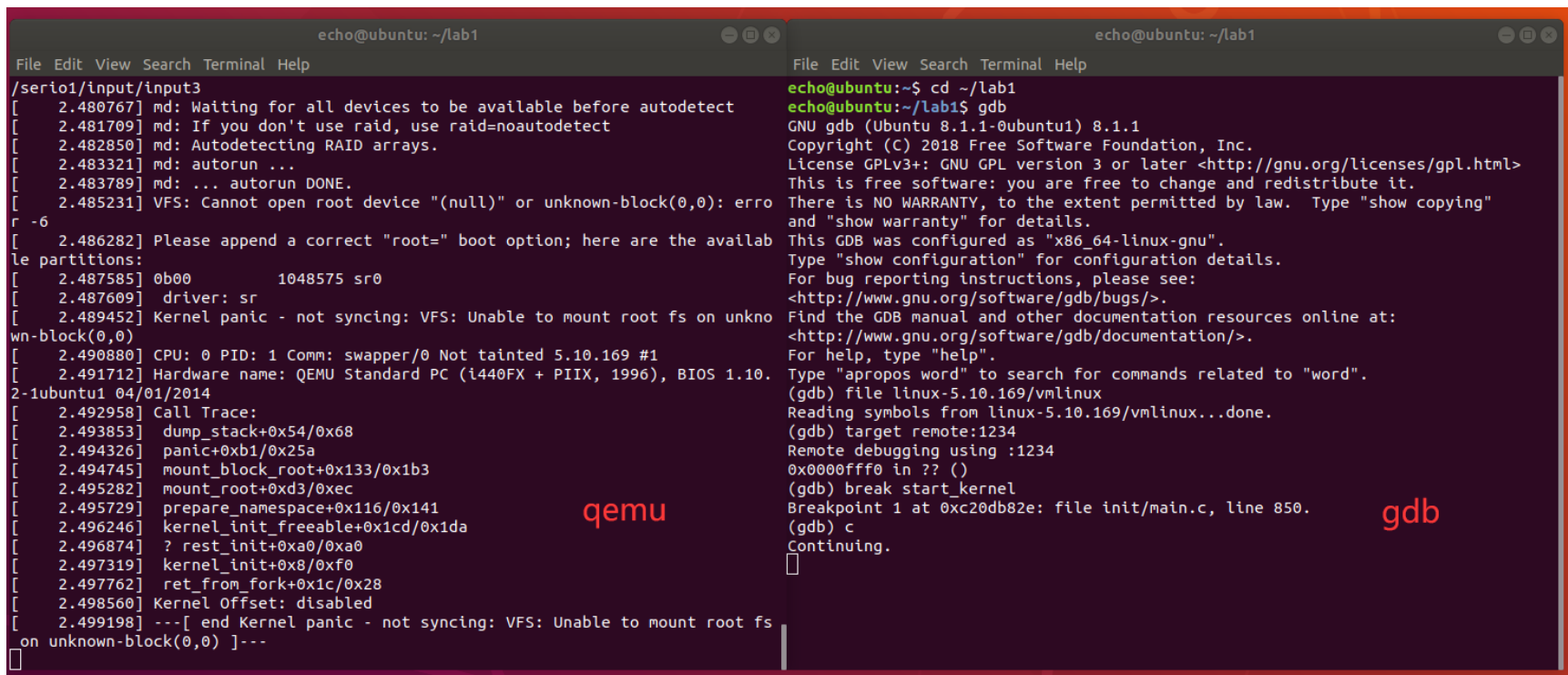
`qemu-system-i386 -kernel linux-5.10.169/arch/x86/boot/bzImage -s -S -append "console=ttyS0" -nographic`

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。

- gdb调试。
 - 不要关闭qemu所在的Terminal。在另外一个Terminal下启动gdb。 `gdb`
 - 在gdb下，加载符号表。 `file linux-5.10.169/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`

实验一 启动内核并调试

- 在继续执行后，最终qemu的输出如下，在qemu虚拟机里运行的Linux系统能成功启动，并且最终以Kernel panic宣告结束。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。



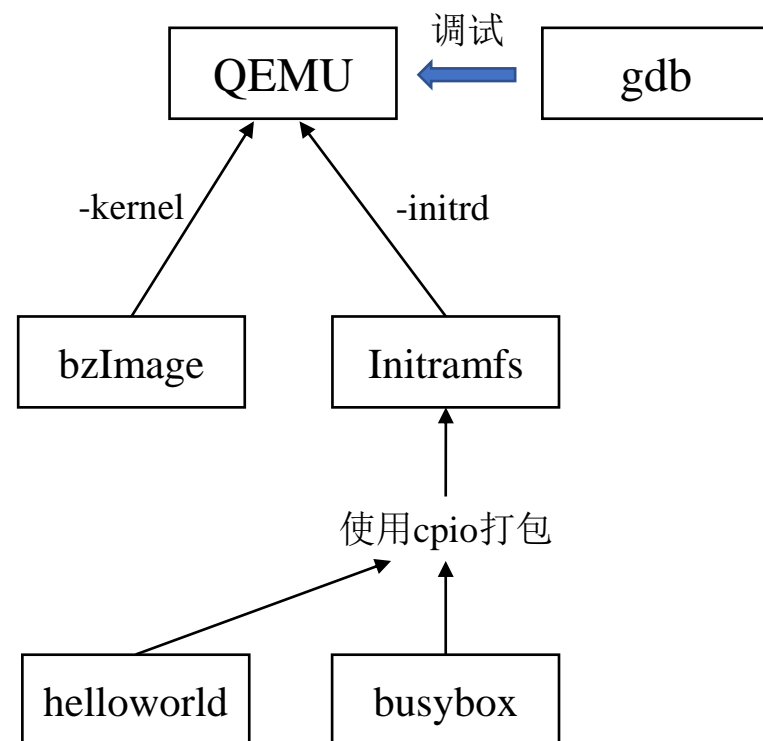
The image shows two terminal windows side-by-side. The left window is QEMU, and the right window is GDB.

QEMU Terminal Output:

```
File Edit View Search Terminal Help
/serio1/input/input3
[ 2.480767] md: Waiting for all devices to be available before autodetect
[ 2.481709] md: If you don't use raid, use raid=noautodetect
[ 2.482850] md: Autodetecting RAID arrays.
[ 2.483321] md: autorun ...
[ 2.483789] md: ... autorun DONE.
[ 2.485231] VFS: Cannot open root device "(null)" or unknown-block(0,0): error -6
[ 2.486282] Please append a correct "root=" boot option; here are the available partitions:
[ 2.487585] 0b00          1048575 sr0
[ 2.487609] driver: sr
[ 2.489452] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.10.169 #1
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
Call Trace:
[ 2.492958] dump_stack+0x54/0x68
[ 2.493853] panic+0xb1/0x25a
[ 2.494326] mount_block_root+0x133/0x1b3
[ 2.494745] mount_root+0xd3/0xec
[ 2.495282] prepare_namespace+0x116/0x141
[ 2.495729] kernel_init_freeable+0x1cd/0x1da
[ 2.496246] ? rest_init+0xa0/0xa0
[ 2.496874] kernel_init+0x8/0xf0
[ 2.497319] ret_from_fork+0x1c/0x28
[ 2.497762] Kernel Offset: disabled
[ 2.498560] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

GDB Terminal Output:

```
File Edit View Search Terminal Help
echo@ubuntu: ~/lab1
echo@ubuntu:~$ cd ~/lab1
echo@ubuntu:~/lab1$ 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 linux-5.10.169/vmlinux
Reading symbols from linux-5.10.169/vmlinux...done.
(gdb) target remote:1234
Remote debugging using :1234
0x0000ffff in ?? ()
(gdb) break start_kernel
Breakpoint 1 at 0xc20db82e: file init/main.c, line 850.
(gdb) c
Continuing.
[]
```



实验一 制作Initramfs

- 进入文件夹 ~/lab1

```
cd ~/lab1
```

- 编写 Hello World 程序

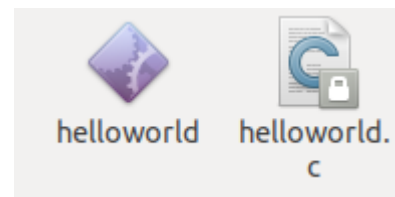
在前面调试内核中，我们已经准备了一个Linux启动环境，但是缺少initramfs。我们可以做一个最简单的Hello World initramfs，来直观地理解initramfs，Hello World程序如下。

```
#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
```

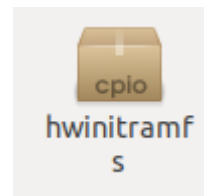


实验一 制作Initramfs

- 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-5.10.169/arch/x86/boot/bzImage -initrd hwinitramfs -s -S -append`

`"console=ttyS0 rdinit=helloworld" -nographic`

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

Two terminal windows are shown side-by-side. The left window, titled 'echo@ubuntu: ~/lab1', displays the output of a kernel boot process. It shows various initialization steps, including loading certificates, enabling netconsole, and loading the kernel image. The final line of the boot process is 'lab1: Hello World', which is highlighted with a red box. The right window, also titled 'echo@ubuntu: ~/lab1', shows the output of a GDB session. It displays the GDB version (8.1.1), copyright information, and the configuration details. The GDB session is in the middle of debugging the kernel, with the target set to 'remote:1234' and the breakpoint set to 'start_kernel'.

实验一 编译并启动Busybox

- 进入文件夹 ~/lab1

```
cd ~/lab1
```

- 下载 Busybox 并解压

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

- 编译 busybox

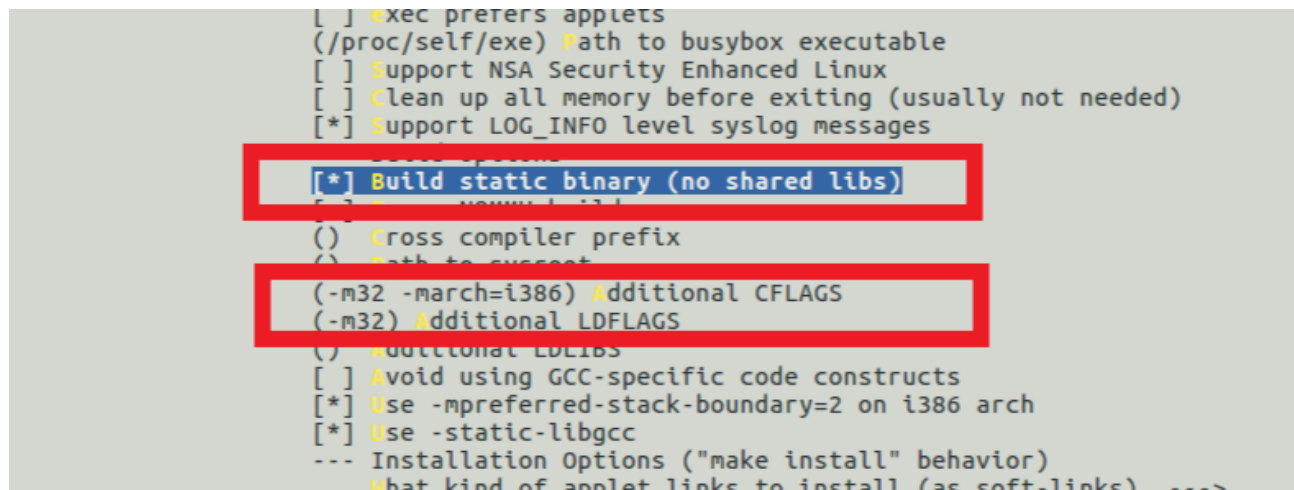
```
make defconfig
```

```
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，编译Busybox

```
make -j8
```

```
make install
```



实验一 编译并启动Busybox

- 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
```

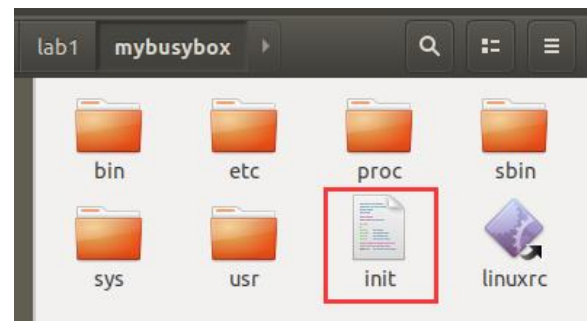
```
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
```

```
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gedit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
```



实验一 编译并启动Busybox

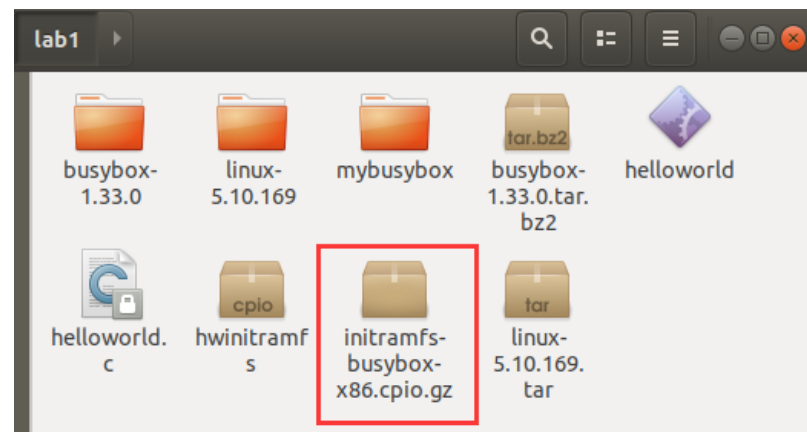
- 制作 Initramfs

- 加上执行权限

```
chmod u+x init
```

- 将x86-busybox下面的内容打包归档成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-x86.cpio.gz
```



- 加载busybox

```
cd ~/lab1
```

```
qemu-system-i386 -kernel linux-5.10.169/arch/x86/boot/bzImage -initrd initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0"
```

然后使用ls命令即可看到当前文件夹

```
/ # ls
bin      etc      linuxrc  root     sys
dev      init     proc     sbin     usr
/ #
```