

《操作系统原理》实验报告			
实验名称	Linux 内核编译	实验序号	7
实验日期	2023/5/14	实验人	盖乐
<p>一、实验题目</p> <p>下载、编译内核源代码[1][2] 启动测试所编译出来的内核 使用 Clang 编译内核* 成功配置 Linux Kernel 静态分析工具[3]</p>			
<p>二、相关原理与知识</p> <p>1. 查看 Linux 内核版本</p> <p>Usage: uname [OPTION]...</p> <p>Print certain system information. With no OPTION, same as -s.</p> <p>-a, --all print all information, in the following order, except omit -p and -i if unknown:</p> <p>-s, --kernel-name print the kernel name</p> <p>-n, --nodename print the network node hostname</p> <p>-r, --kernel-release print the kernel release</p> <p>-v, --kernel-version print the kernel version</p> <p>-m, --machine print the machine hardware name</p> <p>-p, --processor print the processor type (non-portable)</p> <p>-i, --hardware-platform print the hardware platform (non-portable)</p> <p>-o, --operating-system print the operating system</p> <p>    --help display this help and exit</p> <p>    --version output version information and exit</p> <p>2. Linux 内核文件</p> <p>几种 linux 内核文件的区别:</p> <p>1) vmlinux</p> <p>编译出来的最原始的内核文件，未压缩。</p> <p>2) zImage</p> <p>是 vmlinu x 经过 gzip 压缩后的文件。</p> <p>3) bzImage bz 表示"big zImage", 不是用 bzip2 压缩的。两者的不同之处在于，zImage 解压缩内核到低端内存(第一个 640K)，bzImage 解压缩内核到高端内存(1M 以上)。如果内核比较小，那么采用 zImage 或 bzImage 都行，如果比较大应该用 bzImage。</p>			

#### 4) ulImage

U-boot 专用的映像文件，它是在 zImage 之前加上一个长度为 0x40 的 tag。

#### 5) vmlinuz

是 bzImage/zImage 文件的拷贝或指向 bzImage/zImage 的链接。

#### 6) initrd

是"initial ramdisk"的简写。一般被用来临时的引导硬件到实际内核 vmlinuz 能够接管并继续引导的状态。

### 3. Linux 下文件压缩、解压缩指令

\*.Z // compress 程序压缩产生的文件(现在很少使用)

\*.gz // gzip 程序压缩产生的文件

\*.bz2 // bzip2 程序压缩产生的文件

\*.zip // zip 压缩文件

\*.rar // rar 压缩文件

\*.7z // 7- zip 压缩文件

\*.tar // tar 程序打包产生的文件

\*.tar.gz // 由 tar 程序打包并由 gzip 程序压缩产生的文件

\*.tar.bz2 // 由 tar 程序打包并由 bzip2 程序压缩产生的文件

gzip:

gzip 可以压缩产生后缀为.gz 的压缩文件，也可以用于解压 gzip、compress 等程序压缩产生的文件。不带任何选项和参数使用 gzip 或只带有参数“-”时，gzip 从标准输入读取输入，并在标准输出输出压缩结果。

基础格式: gzip [Options] file1 file2 file3

指令选项: (默认功能为压缩)

-c

//将输出写至标准输出，并保持原文件不变

-d

//进行解压操作

-v

//输出压缩/解压的文件名和压缩比等信息

-digit

//digit 部分为数字(1-9)，代表压缩速度，digit 越小，则压缩速度越快，但压缩效果越差，digit 越大，则压缩速度越慢，压缩效果越好。默认为 6。

## bzip2:

bzip2 是采用更好压缩算法的压缩程序，一般可以提供较之 gzip 更好的压缩效果。其具有与 gzip 相似的指令选项，压缩产生.bz2 后缀的压缩文件。

基础格式: bzip2 [Options] file1 file2 file3

指令选项: (默认功能为压缩)

-c

//将输出写至标准输出

-d

//进行解压操作

-v

//输出压缩/解压的文件名和压缩比等信息

-k

//在压缩/解压过程中保留原文件

-digit

//digit 部分为数字(1-9)，代表压缩速度，digit 越小，则压缩速度越快，但压缩效果越差，digit 越大，则压缩速度越慢，压缩效果越好。默认为 6。

## 打包指令—tar:

gzip 或 bzip2 带有多个文件作为参数时，执行的操作是将各个文件独立压缩，而不是将其放在一起进行压缩。这样就无法产生类似于 windows 环境下的文件夹打包压缩的效果。(gzip 与 bzip2 也可以使用文件夹作为参数，使用 -f 选项，但也是将其中的每个文件独立压缩)。为了实现打包压缩的效果，可以使用命令 tar 进行文件的打包操作(archive)，再进行压缩。

基本格式: tar [options] file\_archive // 注意 tar 的第一参数必须为命令选项，即不能直接接待处理文件

常用命令参数:

//指定 tar 进行的操作，以下三个选项不能出现在同一条命令中

-c

//小写,创建一个新的打包文件(archive)

-x

//对打包文件(archive)进行解压操作

-t

//查看打包文件(archive)的内容,主要是构成打包文件(archive)的文件名

//指定支持的压缩/解压方式，操作取决于前面的参数，若为创建(-C),则进行压缩，若为解压(-x),则进行解压，不加下列参数时，则为单纯的打包操作(而不

进行压缩), 产生的后缀文件为.tar

**-z**

//使用 gzip 进行压缩/解压, 一般使用. tar . gz 后缀

**-j**

//使用 bzip2 进行压缩/解压, 一般使用. tar .bz2 后缀

//指定 tar 指令使用的文件, 若没有压缩操作, 则以. tar 作为后缀

**-f filename** // -f 后面接操作使用的文件, 用空格隔开, 且中间不能有其他参数, 推荐放在参数集最后或单独作为参数

//文件作用取决于前面的参数, 若为创建(-c),则-f 后为创建的文件的名称(路径), 若为(-x/t),则-f 后为待解压/查看的打包压缩文件名

//其他辅助选项

**-v**

//详细显示正在处理的文件名

**-C Dir**

//大写,将解压文件放置在-C 指定的目录下

**-p**(小写)

//保留文件的权限和属性, 在备份文件时较有用

**-P**(大写)

//保留原文件的绝对路径, 即不会拿掉文件路径开始的根目录, 则在还原时会覆盖对应路径上的内容

**--exclude=file** //排除不进行打包的文件

**xz:**

解压 tar.xz 文件:先 xz -d xxx.tar.xz 将 xxx.tar.xz 解压成 xxx.tar 然后,再用 tar xvf xxx. tar 来解包。

#### 4. qemu 相关知识

QEMU 是一种通用的开源计算机仿真器和虚拟机。QEMU 共有两种操作模式  
全系统仿真: 能够在任意支持的架构上为任何机器运行一个完整的操作系统用户  
模式仿真: 能够在任意支持的架构上为另一个 Linux/BSD 运行程序同时当进行虚拟化时, QEMU 也可以以接近本机的性能运行 KVM 或者 Xen。

安装:

\$ sudo apt install qemu

安装之后查看会发现有以下应用程序:

```
ouritsusei@ubuntu:~/Desktop$ qemu  
qemu-img qemu-nbd qemu-system-x86_64  
qemu-io qemu-pr-helper  
qemu-make-debian-root qemu-system-i386
```

其中, `qemu-system-x86_64` 用于模拟 64 位 intel 架构 CPU, `qemu-system-i386` 模拟 32 位 intel 架构 CPU, `qemu-system-arm` 模拟 ARM 架构(32 位), `qemu-system-aarch64` 模拟 ARM 架构(64 位), 等等。

## 5. Busybox

BusyBox 是一个集成了三百多个最常用 Linux 命令和工具的软件。BusyBox 包含了一些简单的工具, 例如 `ls`、`cat` 和 `echo` 等等, 还包含了一些更大、更复杂的工具, 例如 `grep`、`find`、`mount` 以及 `telnet`。有些人将 BusyBox 称为 Linux 工具里的瑞士军刀。简单的说 BusyBox 就好像是个大工具箱, 它集成压缩了 Linux 的许多工具和命令, 也包含了 Linux 系统的自带的 shell。我们在 qemu 中测试内核时, 会将它封装到 `initramfs` 使用。

## 6. Initramfs

Linux 系统启动时使用 `initramfs` (initram file system), `initramfs` 可以在启动早期提供一个用户态环境, 借助它可以完成一些内核在启动阶段不易完成的工作。当然 `initramfs` 是可选的, Linux 中的内核编译选项默认开启 `initrd`。在下面的示例情况中你可能要考虑用 `initramfs`。

- 加载模块, 比如第三方 driver
- 定制化启动过程 (比如打印 welcome message 等)
- 制作一个非常小的 rescue shell
- 任何 kernel 不能做的, 但在用户态可以做的 (比如执行某些命令)

一个 `initramfs` 至少要包含一个文件, 文件名为 `/init`。内核将这个文件执行起来的进程作为 main init 进程。当内核挂载 `initramfs` 后, 文件系统的根分区还没有被 mount, 这意味着你不能访问文件系统中的任何文件。如果你需要一个 shell, 必须把 shell 打包到 `initramfs` 中, 如果你需要一个简单的工具, 比如 `ls`, 你也必须把它和它依赖的库或者模块打包到 `initramfs` 中。总之, `initramfs` 是一个完全独立运行的体系。

另外 `initramfs` 打包的时候, 要求打包成压缩的 `cpio` 档案。`cpio` 档案可以嵌入到内核 image 中, 也可以作为一个独立的文件在启动的过程中被 GRUB load。

### 三、实验过程

#### 1. 安装依赖库

```
gaile@gaile-virtual-machine: ~  
(base) gaile@gaile-virtual-machine:~$ sudo apt-get update  
命中:1 http://cn.archive.ubuntu.com/ubuntu jammy InRelease  
命中:2 http://cn.archive.ubuntu.com/ubuntu jammy-updates InRelease  
命中:3 http://cn.archive.ubuntu.com/ubuntu jammy-backports InRelease  
命中:4 https://ppa.launchpadcontent.net/lutris-team/lutris/ubuntu jammy InRelease  
命中:5 http://security.ubuntu.com/ubuntu jammy-security InRelease  
正在读取软件包列表... 完成  
(base) gaile@gaile-virtual-machine:~$ sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils qemu flex libncurses5-dev fakeroot build-essential ncurses-dev xz-utils libssl-dev bc bison libglib2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev libelf-dev  
正在读取软件包列表... 完成  
正在分析软件包的依赖关系树... 完成  
正在读取状态信息... 完成  
注意，选中 'libncurses-dev' 而非 'ncurses-dev'  
注意，选中 'libncurses-dev' 而非 'ncurses-dev'  
bc 已经是最新版 (1.07.1-3build1)。  
bc 已设置为手动安装。  
build-essential 已经是最新版 (12.9ubuntu3)。  
build-essential 已设置为手动安装。  
fakeroot 已经是最新版 (1.28-1ubuntu1)。  
fakeroot 已设置为手动安装。  
libncurses-dev 已经是最新版 (6.3-2)。
```

#### 2. 获取内核源码

首先查看现有 Linux 系统的内核版本


```
(base) gaile@gaile-virtual-machine:~$ uname -srm  
Linux 5.19.0-40-generic x86_64
```

发现为 Linux 5.19.0-40-generic x86\_64; 查看现在最新内核版本为 6.3.2, 进行更新操作。



Protocol	Location
HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>
GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>
RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>

Latest Release

6.3.2 

mainline:	6.4-rc1	2023-05-07	[tarball]	[patch]	[view diff]	[browse]	
stable:	6.3.2	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
stable:	6.2.15	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	6.1.28	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	5.15.111	2023-05-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	5.10.179	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	5.4.242	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	4.19.282	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	4.14.314	2023-04-26	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
linux-next:	next-20230512	2023-05-12					[browse]

### 获取内核源码

```
(base) gaile@gaile-virtual-machine:~$ wget https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.3.2.tar.xz
--2023-05-14 17:29:45-- https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.3.2.tar.xz
正在解析主机 www.kernel.org (www.kernel.org)... 145.40.73.55, 2604:1380:40e1:4800::1
正在连接 www.kernel.org (www.kernel.org)|145.40.73.55|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 301 Moved Permanently
位置: https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/linux-6.3.2.tar.xz [跟随至新的 URL]
--2023-05-14 17:29:46-- https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/linux-6.3.2.tar.xz
正在解析主机 mirrors.edge.kernel.org (mirrors.edge.kernel.org)... 147.75.80.249, 2604:1380:4601:e00::3
正在连接 mirrors.edge.kernel.org (mirrors.edge.kernel.org)|147.75.80.249|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 136908324 (131M) [application/x-xz]
正在保存至: 'linux-6.3.2.tar.xz'

linux-6.3.2.tar.xz          11%[=====>
linulinux-6.3.2.tar.xz  linulinux-linulilinux-6.3.2.lilili
linux-6.3.2.tar.xz          24%[=====> ] 31.80M  530KB/s  剩余 3m 0s ^
linux-6.3.2.tar.xz          32%[=====> ] 43.01M  578KB/s  剩余 2m 40s
```

### 3. 解压内核源码并配置编译选项

```
(base) gaile@gaile-virtual-machine:~$ xz -d linux-6.3.2.tar.xz
```

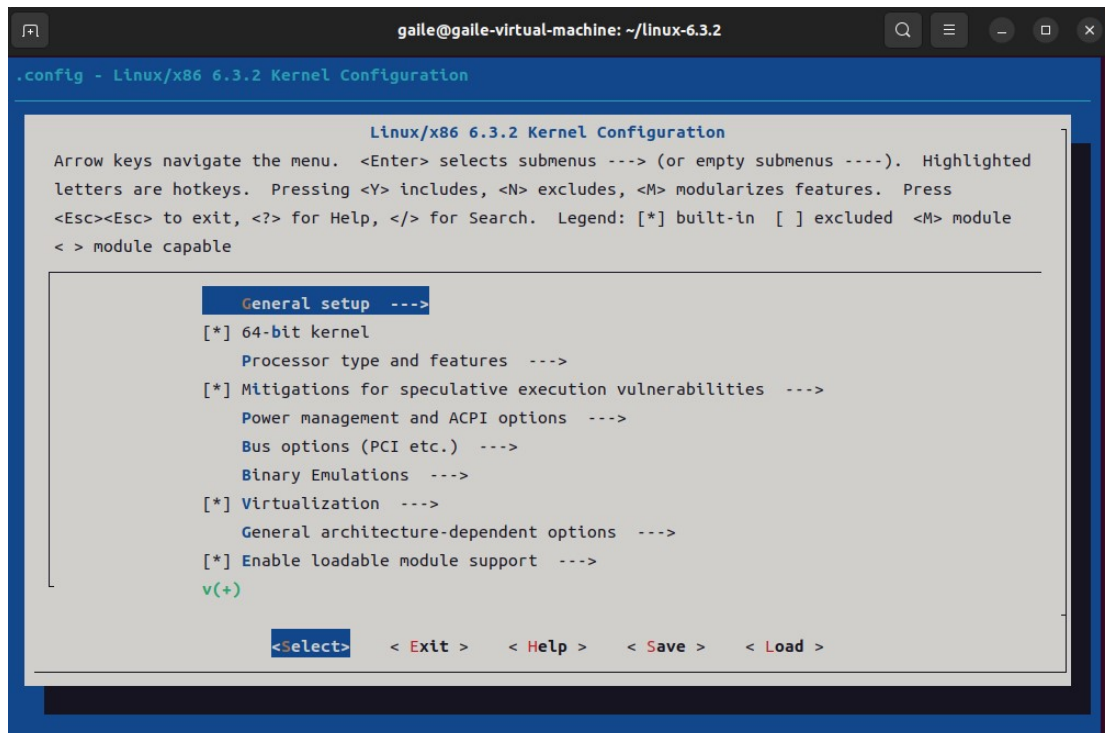
```
(base) gaile@gaile-virtual-machine:~$ tar xvf linux-6.3.2.tar
```

```
(base) gaile@gaile-virtual-machine:~$ ls
公共的  下载      Anaconda3-2023.03-Linux-x86_64.sh  license_29403_0.dat  snap
模板    音乐      ASRT                                linux-6.3.2          Study
视频    桌面      'Clash for Windows-0.20.10-x64-linux' linux-6.3.2.tar      v2rayL
图片    ABY       Code                                OTEExtension
文档    anaconda3 data                                Qv2ary
```

```
(base) gaile@gaile-virtual-machine:~/linux-6.3.2$ ls
arch  COPYING  Documentation  include  ipc      kernel  MAINTAINERS  net      samples  sound  virt
block CREDITS  drivers        init     Kbuild  lib     Makefile     README  scripts  tools
certs crypto   fs             io_uring Kconfig  LICENSES  mm          rust    security  usr
```

在正式编译内核之前，我们首先必须配置需要包含哪些模块。使用 `cp` 命令，将当前内核的配置文件拷贝到当前文件夹，然后使用可靠的 `menuconfig` 命令来做任何必要的更改。使用如下命令来完成：

```
(base) gaile@gaile-virtual-machine:~/linux-6.3.2$ make menuconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/kconfig/mconf.o
HOSTCC  scripts/kconfig/lxdialog/checklist.o
HOSTCC  scripts/kconfig/lxdialog/inputbox.o
HOSTCC  scripts/kconfig/lxdialog/menubox.o
HOSTCC  scripts/kconfig/lxdialog/textbox.o
HOSTCC  scripts/kconfig/lxdialog/util.o
HOSTCC  scripts/kconfig/lxdialog/yesno.o
HOSTCC  scripts/kconfig/confdata.o
HOSTCC  scripts/kconfig/expr.o
LEX      scripts/kconfig/lexer.lex.c
YACC     scripts/kconfig/parser.tab.[ch]
```



由于替换内核所需的存储空间较大，所以先查看是否有充足的存储空间



```
(base) gaile@gaile-virtual-machine:~/linux-6.3.2$ df -h
```

文件系统	大小	已用	可用	已用%	挂载点
tmpfs	792M	2.2M	790M	1%	/run
/dev/sda3	59G	50G	6.5G	89%	/
tmpfs	3.9G	4.2M	3.9G	1%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
/dev/sda2	512M	6.1M	506M	2%	/boot/efi
tmpfs	792M	2.4M	790M	1%	/run/user/1000
/dev/sr0	3.6G	3.6G	0	100%	/media/gaile/Ubuntu 22.04.1 LTS amd64

发现只有不到 10G，所以进行存储空间扩容操作

取消(C)
调整卷大小
调整大小(R)

调整文件系统大小可能会导致数据丢失。建议先备份您的数据。如果有许多数据需要移动，调整大小的操作将花费更长的时间。最小大小根据当前的内容来计算。保留额外的文件系统可用空间以快速可靠地工作。

当前大小

最小大小

分区大小(S)
74.6
GB

后面的可用空间(F)
0.0
GB

差别(D)
10.7
GB

再次查看可用存储空间

```
gaile@gaile-virtual-machine: ~
```

```
(base) gaile@gaile-virtual-machine:~$ df -h
```

文件系统	大小	已用	可用	已用%	挂载点
tmpfs	792M	2.2M	790M	1%	/run
/dev/sda3	69G	50G	17G	76%	/
tmpfs	3.9G	4.4M	3.9G	1%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
/dev/sda2	512M	6.1M	506M	2%	/boot/efi
tmpfs	792M	2.4M	790M	1%	/run/user/1000
/dev/sr0	3.6G	3.6G	0	100%	/media/gaile/Ubuntu 22.04.1 LTS amd64

#### 4. 编译

由于此虚拟机只有一个核，所以采取 make 指令直接编译内核

```
(base) gaile@gaile-virtual-machine:~$ cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 154
model name     : 12th Gen Intel(R) Core(TM) i7-12700H
stepping       : 3
microcode      : 0x413
cpu MHz        : 2687.998
cache size     : 24576 KB
physical id    : 0
siblings       : 12
core id        : 0
cpu cores      : 12
apicid         : 0
```

```
SYSHDR arch/x86/include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_64.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_x32.h
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
SYSHDR arch/x86/include/generated/asm/unistd_32_ia32.h
SYSHDR arch/x86/include/generated/asm/unistd_64_x32.h
SYSTBL arch/x86/include/generated/asm/syscalls_64.h
SYSTBL arch/x86/include/generated/asm/syscalls_x32.h
HYPERCALLS arch/x86/include/generated/asm/xen-hypercalls.h
```

make 指令运行完后，下一步进行编译和安装内核模块

```
(base) gaile@gaile-virtual-machine:~$ make modules_install
make install
```

编译过程中报错：

```
make[1]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'. Stop.
make: *** [Makefile:1831: certs] Error 2
```

经网上查阅资料得出这个时候需要在 Makefile 文件中：

KBUILD\_CFLAGS 的尾部添加选项 -fno-pic

CC\_USING\_FENTRY 项添加 -fno-pic

以及在 .config 文件中找到这一项，等于号后面的值改为 ""

```
CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem"
CONFIG_HAVE_KVM=y
CONFIG_HAVE_KVM_IRQCHIP=y
CONFIG_HAVE_KVM_IRQFD=y
CONFIG_HAVE_KVM_IRQ_ROUTING=y
CONFIG_HAVE_KVM_EVENTFD=y
CONFIG_KVM_APIC_ARCHITECTURE=y
CONFIG_KVM_MMIO=y
CONFIG_KVM_ASYNC_PF=y
CONFIG_HAVE_KVM_MSI=y
CONFIG_HAVE_KVM_CPU_RELAX_INTERCEPT=y
CONFIG_KVM_VFIO=y
/CONFIG_SYSTEM_TRUSTED_KEYS
```

完成之后会出现如下信息：

```
Kernel: arch/x86/boot/bzImage is ready (#1)
```

## 5. 使用 qemu 测试

接下来使用 qemu 对编译出来的内核进行测试，下载 busybox

```
(base) gaille@gaille-virtual-machine:~$ wget https://busybox.net/downloads/busybox-1.35.0.tar.bz2
--2023-05-14 18:08:12-- https://busybox.net/downloads/busybox-1.35.0.tar.bz2
正在解析主机 busybox.net (busybox.net)... 140.211.167.122
正在连接 busybox.net (busybox.net)|140.211.167.122|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 2480624 (2.4M) [application/x-bzip2]
正在保存至：‘busybox-1.35.0.tar.bz2’

busybox-1.35.0.tar.  1%[          ] 31.04K  ---KB/s  剩余 6m 28s^
busybox-1.35.0.tar.  1%[          ] 31.04K  ---KB/s  剩余 11m 21s
```

解压

```
$ tar -jxvf busybox-1.33.0.tar.bz2
```

## 编译 busybox 源码

进入配置界面

```
$ make menuconfig
```

勾选 Settings ---> Build static binary file (no shared lib)(若是不勾选则需要单独配置 lib，比较麻烦)

接下来就是编译了，速度会比编译内核快很多

```
$ make install
```

```
./_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.
```

编译完成后会生成一个\_install 目录，接下来我们将会用它来构建我们的磁盘镜像

applets	busybox_unstripped.map	e2fsprogs	_install	Makefile	NOFORK_NOEXEC.lst	selinux
applets_sh	busybox_unstripped.out	editors	INSTALL	Makefile.custom	NOFORK_NOEXEC.sh	shell
arch	Config.in	examples	klirc-utils	Makefile.flags	printutils	size_single_applets.sh
archival	configs	findutils	libbb	Makefile.help	procps	sysklogd
AUTHORS	console-tools	include	libpwdgrp	make_single_applets.sh	qemu_multiarch_testing	testsuite
busybox	coreutils	init	LICENSE	miscutils	README	TODO
busybox.links	debianutils	initramfs	loginutils	modutils	runit	TODO_unicode
busybox_unstripped	docs	initramfs.cpio.gz	nailutils	networking	scripts	util-linux

Linux 启动阶段，boot loader 加载完内核文件 vmlinuz 之后，便开始挂载磁盘根文件系统。挂载操作需要磁盘驱动，所以挂载前要先加载驱动。但是驱动位于/lib/modules,不挂载磁盘就访问不到，形成了一个死循环。initramfs 根文件系统就可以解决这个问题，其中包含必要的设备驱动和工具，boot loader 会加载 initramfs 到内存中，内核将其挂载到根目录，然后运行/init 初始化脚本，去挂载真正的磁盘根文件系统。

## 6. 构建磁盘镜像

建立文件系统，初始化文化系统，并进行一些简单的初始化操作

```
$ cd _install
$ mkdir -pv {bin,sbin,etc,proc,sys,home,lib64,lib/x86_64-linux-gnu,usr/{bin,sbin}}
$ touch etc/inittab
$ mkdir etc/init.d
$ touch etc/init.d/rcS
$ chmod +x ./etc/init.d/rcS
```

配置初始化脚本，配置 etc/inittab，写入如下内容：

```
::sysinit:/etc/init.d/rcS
::askfirst:/bin/ash
::ctrlaltdel:/sbin/reboot
::shutdown:/sbin/swapoff -a
::shutdown:/bin/umount -a -r
::restart:/sbin/init
```



在上面的文件中指定了系统初始化脚本，因此接下来配置 `etc/init.d/rcS`，写入如下内容：

```
#!/bin/sh
mount -t proc none /proc
mount -t sys none /sys
/bin/mount -n -t sysfs none /sys
/bin/mount -t ramfs none /dev
/sbin/mdev -s
```

主要是配置各种目录的挂载，也可以在根目录下创建 `init` 文件，写入如下内容：

```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
mount -t devtmpfs devtmpfs /dev

exec 0</dev/console
exec 1>/dev/console
exec 2>/dev/console

echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
setsid cttyhack setuidgid 1000 sh

umount /proc
umount /sys
poweroff -d 0 -f
```

添加可执行权限：

```
$ chmod +x ./init
```

打包文件系统为镜像文件，使用如下命令打包文件系统

```
$ find . | cpio -o --format=newc > ../../rootfs.cpio
```

## 7. 使用 qemu 运行内核

配置启动脚本，将先前的 `bzImage` 和 `rootfs.cpio` 放到同一个目录下

接下来编写启动脚本

```
$ touch boot.sh
```



写入如下内容：

```
#!/bin/sh
qemu-system-x86_64 \
    -m 128M \
    -kernel ./bzImage \
    -initrd ./rootfs.cpio \
    -monitor /dev/null \
    -append "root=/dev/ram rdinit=/sbin/init console=ttyS0 oops=panic panic=1 loglevel=3 quiet nokaslr" \
    -cpu kvm64,+smep \
    -smp cores=2,threads=1 \
    -netdev user,id=t0, -device e1000,netdev=t0,id=nic0 \
    -nographic \
    -s
```

部分参数说明如下

**-m:** 虚拟机内存大小

**-kernel:** 内存镜像路径

**-initrd:** 磁盘镜像路径

**-append:** 附加参数选项

**nokaslr:** 关闭内核地址随机化，方便我们进行调试

**rdinit:** 指定初始启动进程，/sbin/init 进程会默认以/etc/init.d/rcS 作为启动脚本

**loglevel=3 & quiet:** 不输出 log

**console=ttyS0:** 指定终端为/dev/ttyS0，这样一启动就能进入终端界面

**-monitor:** 将监视器重定向到主机设备/dev/null，这里重定向至 null 主要是防止 CTF 中被人偷了 qemu 拿 flag

**-cpu:** 设置 CPU 安全选项，在这里开启了 smep 保护

**-s:** 相当于-gdb tcp::1234 的简写（也可以直接这么写），后续我们可以通过 gdb 连接本地端口进行调试

运行 boot.sh，成功启动，说明可以进行内核替换。

```
Please press Enter to activate this console.
/ # ls
bin      etc      lib      linuxrc  root     sys
dev      home     lib64    proc     sbin     usr
/ # whoami
root
/ #
```

## 8. 替换内核

安装模块

```
arch/x86/Makefile:154: CONFIG_X86_X32 enabled but no binutils support
INSTALL /lib/modules/5.17.9/kernel/arch/x86/crypto/aegis128-aesni.ko
SIGN /lib/modules/5.17.9/kernel/arch/x86/crypto/aegis128-aesni.ko
INSTALL /lib/modules/5.17.9/kernel/arch/x86/crypto/aesni-intel.ko
SIGN /lib/modules/5.17.9/kernel/arch/x86/crypto/aesni-intel.ko
INSTALL /lib/modules/5.17.9/kernel/arch/x86/crypto/blake2s-x86_64.ko
```

安装内核

```
arch/x86/Makefile:154: CONFIG_X86_X32 enabled but no binutils support
sh ./arch/x86/boot/install.sh 5.17.9 \
    arch/x86/boot/bzImage System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.17.9 /boot/vmlinuz-5.17.9
update-initramfs: Generating /boot/initrd.img-5.17.9
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.17.9 /boot/vmlinuz-5.17.9
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.17.9 /boot/vmlinuz-5.17.9
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 5.17.9 /boot/vmlinuz-5.17.9
I: /boot/initrd.img.old is now a symlink to initrd.img-5.13.0-44-generic
I: /boot/initrd.img is now a symlink to initrd.img-5.17.9
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.17.9 /boot/vmlinuz-5.17.9
Sourcing file '/etc/default/grub'
Sourcing file '/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.17.9
Found initrd image: /boot/initrd.img-5.17.9
Found linux image: /boot/vmlinuz-5.13.0-44-generic
Found initrd image: /boot/initrd.img-5.13.0-44-generic
Found linux image: /boot/vmlinuz-5.13.0-41-generic
Found initrd image: /boot/initrd.img-5.13.0-41-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

重启后查看内核版本

```
gaile@gaile-virtual-machine: ~
(base) gaile@gaile-virtual-machine:~$ uname -srm
Linux 6.3.2 x86_64
(base) gaile@gaile-virtual-machine:~$
```

## 9. 配置 Crix

先将该静态分析工具下载至本地。

```
(base) gaile@gaile-virtual-machine:~$ git clone https://github.com/umnsec/crix.git
正克隆到 'crix'...
remote: Enumerating objects: 84, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 84 (delta 10), reused 14 (delta 4), pack-reused 53
接收对象中: 100% (84/84), 81.41 KiB | 423.00 KiB/s, 完成.
处理 delta 中: 100% (11/11), 完成.
```

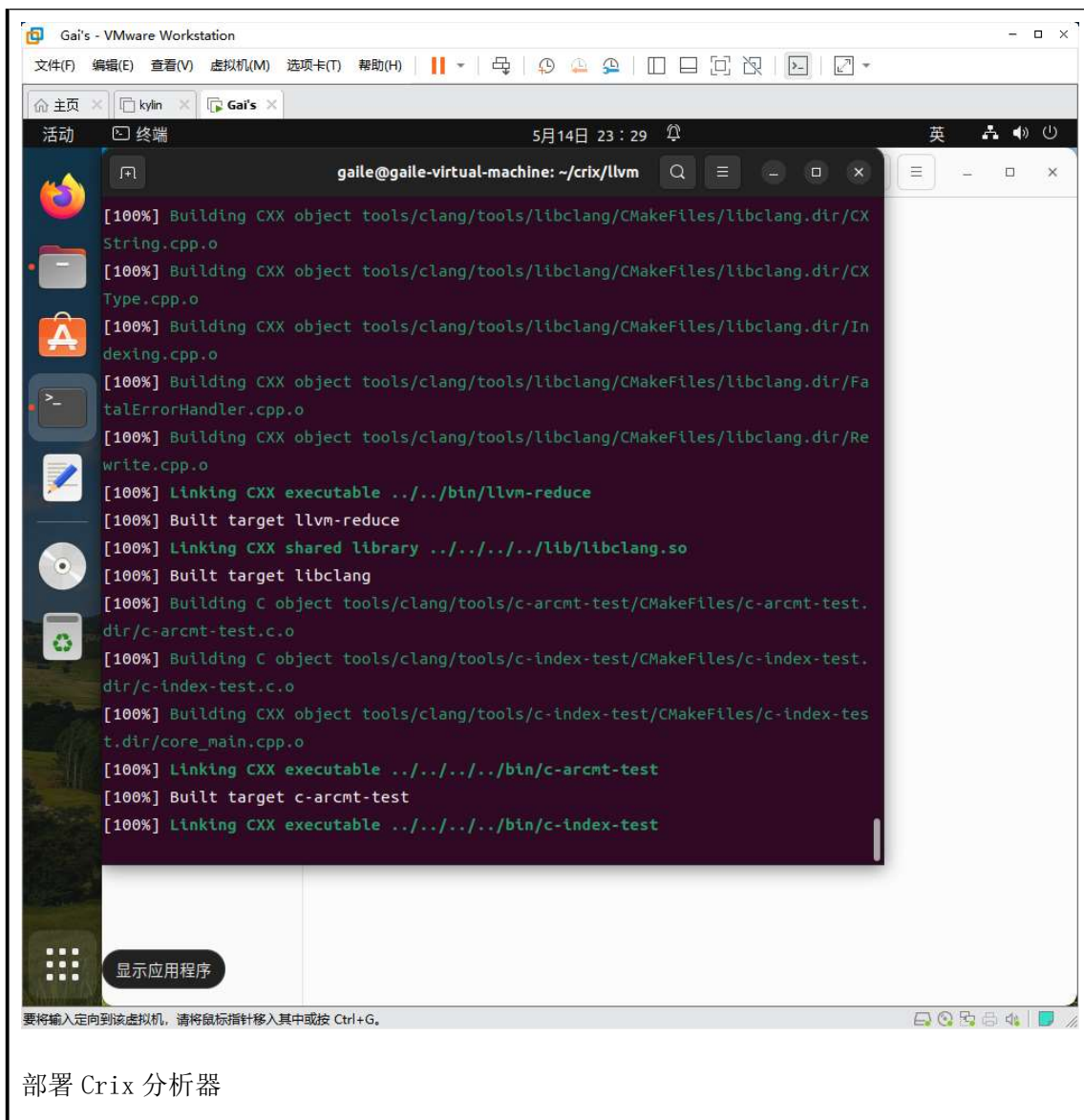
无法安装 Crix analyzer, 是由于缺少 llvm 依赖项

```
(base) gaile@gaile-virtual-machine:~/crix/llvm$ cd ../analyzer
(base) gaile@gaile-virtual-machine:~/crix/analyzer$ make
(mkdir -p /home/gaile/crix/analyzer/build && cd /home/gaile/crix/analyzer/build && PATH=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/bin:/home/gaile/anaconda3/bin:/home/gaile/anaconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin LLVM_TOOLS_BINARY_DIR=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/bin LLVM_LIBRARY_DIRS=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/lib LLVM_INCLUDE_DIRS=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/include CC=clang CXX=clang++ cmake /home/gaile/crix/analyzer/src -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_ASSERTIONS=ON -DCMAKE_CXX_FLAGS_RELEASE="-std=c++14 -fno-rtti -fpic -g" && make -j12)
-- The C compiler identification is Clang 14.0.0
-- The CXX compiler identification is Clang 14.0.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/clang - skipped
-- Detecting C compile features
```

从 github 安装依赖 llvm

```
gaile@gaile-virtual-machine: ~
(base) gaile@gaile-virtual-machine:~$ git clone https://github.com/llvm/llvm-project.git
正克隆到 'llvm-project'...
remote: Enumerating objects: 5394613, done.
remote: Counting objects: 100% (2924/2924), done.
remote: Compressing objects: 100% (429/429), done.
接收对象中: 1% (102822/5394613), 36.59 MiB | 3.88 MiB/sB/s
```

安装 llvm, 安装的时候经常会出现闪退的情况, 莫名其妙闪退, 重启之后莫名其妙就好了, 然后就是长达两小时的部署。



部署 Crux 分析器



```
gaile@gaile-virtual-machine: ~/crix/analyzer
(base) gaile@gaile-virtual-machine:~/crix/analyzer$ make
(mkdir -p /home/gaile/crix/analyzer/build && cd /home/gaile/crix/analyzer/build && PATH=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/bin:/home/gaile/anaconda3/bin:/home/gaile/anaconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin LLVM_TOOLS_BINARY_DIR=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/bin LLVM_LIBRARY_DIRS=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/lib LLVM_INCLUDE_DIRS=/home/gaile/crix/analyzer/../llvm/llvm-project/prefix/include CC=clang CXX=clang++ cmake /home/gaile/crix/analyzer/src -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_ASSERTIONS=ON -DCMAKE_CXX_FLAGS_RELEASE="-std=c++14 -fno-rtti -fpic -g" && make -j12)
-- Found LLVM 14.0.0
-- Using LLVMConfig.cmake in: /usr/lib/llvm-14/cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gaile/crix/analyzer/build
make[1]: 进入目录"/home/gaile/crix/analyzer/build"
make[2]: 进入目录"/home/gaile/crix/analyzer/build"
make[3]: 进入目录"/home/gaile/crix/analyzer/build"
Consolidate compiler generated dependencies of target AnalyzerObj
make[3]: 离开目录"/home/gaile/crix/analyzer/build"
make[3]: 进入目录"/home/gaile/crix/analyzer/build"
[ 5%] Building CXX object lib/CMakeFiles/AnalyzerObj.dir/Common.cc.o
[ 10%] Building CXX object lib/CMakeFiles/AnalyzerObj.dir/DataFlowAnalysis.cc.o
```

运行 Crix 分析器

```
(base) gaile@gaile-virtual-machine:~/crix/analyzer$ # To analyze a single bitcode file, say "test.bc", run:
./build/lib/kanalyzer -sc test.bc
# To analyze a list of bitcode files, put the absolute paths of the bitcode files in a file, say "bc.list", then run:
./build/lib/kanalyzer -mc @bc.list
```

成功运行



```
gaile@gaile-virtual-machine: ~/crux/llvm
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Config/Tar
getMCAs.def
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Config/Asm
Printers.def
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Config/Dis
assemblers.def
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Config/Tar
gets.def
-- Up-to-date: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend
-- Up-to-date: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penMP
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penMP/OMP.inc
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penMP/OMP.h.inc
-- Up-to-date: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penACC
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penACC/ACC.inc
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/include/llvm/Frontend/O
penACC/ACC.h.inc
-- Installing: /home/gaile/crux/llvm/llvm-project/prefix/lib/cmake/llvm/LLVMConf
igExtensions.cmake
```

#### 四、实验结果与分析

查询当前系统的内核版本

```
gaile@gaile-virtual-machine: ~
(base) gaile@gaile-virtual-machine:~$ uname -srm
Linux 6.3.2 x86_64
(base) gaile@gaile-virtual-machine:~$
```

可以看到当前系统的内核版本已被更换为 6.3.2

安装 Clang

```
(base) gaile@gaile-virtual-machine:~$ sudo apt-get install clang
[sudo] gaile 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
将会同时安装下列软件:
  binfmt-support clang-14 lib32gcc-s1 lib32stdc++6 libc6-i386
  libclang-common-14-dev libobjc-11-dev libobjc4 libpfm4 libz3-4 libz3-dev
  llvm-14 llvm-14-dev llvm-14-linker-tools llvm-14-runtime llvm-14-tools
建议安装:
  clang-14-doc llvm-14-doc
下列【新】软件包将被安装:
  binfmt-support clang clang-14 lib32gcc-s1 lib32stdc++6 libc6-i386
  libclang-common-14-dev libobjc-11-dev libobjc4 libpfm4 libz3-4 libz3-dev
  llvm-14 llvm-14-dev llvm-14-linker-tools llvm-14-runtime llvm-14-tools
升级了 0 个软件包，新安装了 17 个软件包，要卸载 0 个软件包，有 23 个软件包未被升级。
需要下载 68.9 MB 的归档。
解压缩后会消耗 448 MB 的额外空间。
您希望继续执行吗？ [Y/n] y
获取:1 http://cn.archive.ubuntu.com/ubuntu jammy/main amd64 binfmt-support amd64
2.2.1-2 [55.8 kB]
```

## 五、问题总结

1. 编译内核模块到一半时提示空间不足，只好先关闭虚拟机，进行磁盘拓展后再重新启动，拓展好磁盘后再继续进行编译。
2. 编译内核时遇到报错：  
第一是因为直接使用的当前配置文件，没有进行修改导致无法找到对应文件；  
第二是由于未安装对应的应用，`sudo apt install` 后成功解决；  
第三是启动内核时出现“loading initial ramdisk”，这是因为 `initrd` 过大导致的，通过上网查找相关资料解决。
3. 在部署 Crix 分析器时命令行莫名其妙闪退，经过多次尝试终于成功部署。

## 六、参考文献

- [1] [https://bugzilla.redhat.com/show\\_bug.cgi?id=1572126](https://bugzilla.redhat.com/show_bug.cgi?id=1572126)
- [2] 启动内核卡在“loading initial ramdisk”解决方案：  
<https://codeantenna.com/a/ZkNULLTebn>
- [3] Why is INSTALL\_MOD\_STRIP not on by default?  
<https://superuser.com/questions/705121/why-is-installmod-strip-not-on-by-default>
- [4] Linux 内核编译  
[https://github.com/arttnba3/XDU-SCE\\_OS-Experiment\\_2021/tree/main/Exp-7](https://github.com/arttnba3/XDU-SCE_OS-Experiment_2021/tree/main/Exp-7)
- [5] <https://fonttian.blog.csdn.net/article/details/103924589>

- [6] <https://blog.csdn.net/FontThrone/article/details/104157859>
- [7] Linux 内核编译 <https://os.51cto.com/article/663841.html>
- [8]如何编译 Linux 内核 <https://zhuanlan.zhihu.com/p/37164435>
- [9] Linux 内核编译与安装 <https://www.linuxprobe.com/linux-kernel-compilation.html>
- [10] Linux 内核编译步骤及配置详解 <https://www.cnblogs.com/xiaocen/p/3717993.html>
- [11]如何编译 Linux 内核 <https://linux.cn/article-9665-1.html>
- [12] Kernel Panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)  
<https://askubuntu.com/questions/41930/kernel-panic-not-syncing-vfs-unable-to-mount-root-fs-on-unknown-block0-0>
- [13] Boot hangs at "loading initial ramdisk"  
<https://bugs.launchpad.net/ubuntu/+source/intel-microcode/+bug/1853369>
- [14] qemu 运行 linux 内核 <https://www.cnblogs.com/tbolp/p/15219547.html>
- [15] Linux 内存管理之环境搭建 <https://luomuxiaoxiao.com/?p=743>
- [16] QEMU 运行 Linux Kernel 环境配置  
<https://www.cxyzjd.com/article/HaoBBNuanMM/106625017>
- [17] QEMU + Busybox 模拟 Linux 内核环境 <https://www.v4ler1an.com/2020/12/qemu/>
- [18]基于 Qemu 搭建 x86\_64 虚拟环境运行 Linux 内核  
[https://blog.csdn.net/fantasy\\_wxe/article/details/108418822](https://blog.csdn.net/fantasy_wxe/article/details/108418822)