**《操作系统》实验报告**

徽标, 公司名称

描述已自动生成

**题目: Linux内核编译与安装**

**学院 计算机学院（国家示范性软件学院）**

**班级 2021211301**

**学号 2021213586**

**姓名 郭栩源**

**2024年 5月**

**目录**

[**第一章 实验概述** 1](#_Toc167483688)

[1.1 实验内容 1](#_Toc167483689)

[1.1.1 编译并安装Linux内核 1](#_Toc167483690)

[1.1.2 分析系统启动时间 1](#_Toc167483691)

[**第二章 实验步骤** 1](#_Toc167483692)

[2.1 编译并安装Linux内核 1](#_Toc167483693)

[2.1.1下载内核源代码 1](#_Toc167483694)

[2.1.2 编译Linux内核和模块 4](#_Toc167483695)

[2.1.3 安装Linux内核和模块 5](#_Toc167483696)

[2.2 分析系统启动时间 6](#_Toc167483697)

[2.2.1测定Linux系统启动时间 6](#_Toc167483698)

[2.2.2分析系统启动时间 7](#_Toc167483699)

[**第三章 实验总结** 8](#_Toc167483700)

# **第一章 实验概述**

## 1.1 实验内容

### 1.1.1 编译并安装Linux内核

作为自由软件，在广大爱好者的支持下，Linux内核版本不断更新。新的内核修订了旧内核的bug，并增加了许多新的特性。如果用户想要使用这些特性，或想根据自身系统量身定制一个更高效、更稳定的内核，就需要重新编译内核。通常，更新的内核会支持更多的硬件，具备更好的进程管理能力，运行速度更快、更稳定，并会修复旧版本中发现的许多漏洞等。经常升级更新的系统内核是Linux使用者的必要操作内容。

具体实验内容如下：

1.确认系统环境：在进行内核编译之前，首先需要确认系统环境。可以使用命令 uname -a 查看当前系统的内核版本和系统架构，以及使用 lsb\_release -a 查看系统发行版本。

2.下载内核源码：从官方网站或者镜像站点下载所需的Linux内核源码包。通常情况下，内核源码会以压缩包的形式提供，如.tar.gz或.tar.xz等。

3.解压源码包：使用合适的解压命令，如 tar -xf 或 tar -xJf 解压下载的内核源码压缩包。

4.配置内核选项：进入解压后的内核源码目录，在命令行中输入 make menuconfig 或 make xconfig 进入内核配置界面。在这个界面中，可以选择所需的内核功能、驱动程序和特性，也可以进行各种配置以定制化内核。

5.编译内核：完成配置后，使用 make 命令编译内核。编译过程可能会花费一些时间，取决于系统配置和内核版本。

6.安装内核：编译完成后，使用 make install 命令安装新的内核。这会将编译好的内核文件复制到系统相应的位置，并更新启动引导程序（如GRUB）的配置文件。

7.更新启动引导程序：如果安装了多个内核版本，需要更新启动引导程序的配置，以便在系统启动时可以选择使用哪个内核版本。

8.验证内核版本：系统重新启动后，使用 uname -r 命令验证当前系统正在使用的内核版本，确认新内核已经成功安装并启动。

### 1.1.2 分析系统启动时间

systemd-analyze 是一个系统工具，用于分析和调试 systemd 启动过程的性能问题。它提供了多种功能，帮助用户了解系统启动的各个阶段、服务加载时间以及整个系统的启动时间。

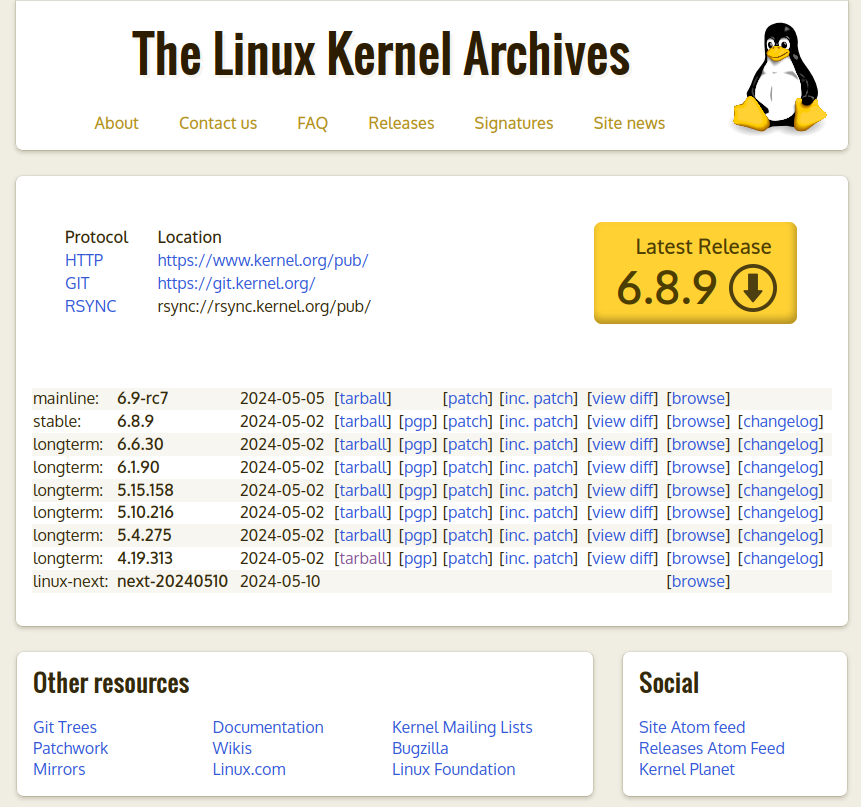
使用systemd-analyse来显示系统启动时间，使用systemd-analyze critical-chain显示启动过程中服务的依赖关系及其启动时间。

# **第二章 实验步骤**

## 2.1 编译并安装Linux内核

### 2.1.1下载内核源代码

Linux操作系统有GNU通用公共许可证(GPL)保护,故其内核源代码是完全开放的。现在很多Linux的网站都提供内核代码的下载。推荐使用Linux的官方网站(http://www. kernel.org),在这里可以找到所有的内核版本,如下图所示：



本次实验选择内核版本为4.19。

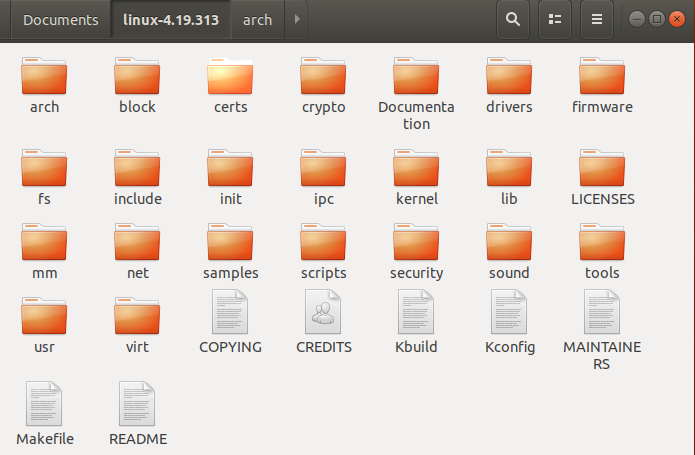
使用命令

# sudo tar -C /usr/src -xf linux-4.19.313.tar.gz

将下载的tar文件解包至/usr/src文件夹下：



可以看待源程序树的最上层有一些目录：



Arch子目录包括所有和体系结构相关的核心代码。它还有更深的子目录,每一个代表一种支持的体系结构,例如i386和alpha。

Include子目录包括编译核心所需要的大部分include文件。它也有更深的子目录,分别对应所支持的各个体系结构。Include/asm是这个体系结构所需要的真实的include目录的软链接,例如include/asm-i386。为了改变体系结构，你需要编辑核心的makefile,重新运行Linux的核心配置程序。

Init子目录包含核心的初始化代码，这是研究核心如何工作的一个非常好的起点。

Mm子目录包括所有的内存管理代码。和体系结构相关的内存管理代码位于 arch/\*/mm/,例如 arch/i386/mm/fault.c

Drivers系统所有的设备驱动程序。它们被划分成设备驱动程序类，例如 block。

Ipc子目录包含核心的进程间通讯的代码。

Modules这只是一个用来存放建立好的模块的目录。

Fs所有的文件系统代码。被划分成子目录,每一个支持的文件系统一个子目录,例如vfat和 ext2。

Kernel 主要的核心代码。同样，和体系相关的核心代码放在arch/\*/kernel。 Net核心的网络代码。

Lib子目录放置核心的库代码。和体系结构相关的库代码放在arch/\*/lib。 Scripts子目录包含脚本(例如awk和tk脚本),用于配置核心。

Memory Management(内存管理)代码大多存在mm下,但是和体系结构相关的代码在arch/\*/mm。Page fault处理代码在mm/memory.c中,内存映射和页缓存代码在mm/filemap.c中。Buffer cache在mm/buffer.c中实现,交换缓存在 mm/swap\_state.c和mm/swapfile.c中。

Kernel:大部分相对通用的代码存在kernel下,和体系结构相关的代码在 arch/\* /kernel。调度程序在kernel/sched.c,fork代码在kernel/fork.c。bottom half 处理代码在include/linux/interrupt.h。task\_struct数据结构可以在include/ linux/sched.h中找到。

Pci:Pci伪驱动程序在drivers/pci/pci.c,系统范围的定义在include/linux/ pci.h。每一种体系结构都有一些特殊的PciBIOS代码，Alpha axp的位于arch/ alpha/kernel/bios32.c。

Interprocess Communication:全部在Ipc目录。所有系统VIPC对象都包括 ipc\_perm 数据结构,可以在include/linux/ipc.h中找到。系统V消息在ipc/msg. c中实现,共享内存在ipc/shm.c中,信号灯在ipc/sem.c,管道在ipc/pipe.c中实现。

Interrupt Handling:核心的中断处理代码几乎都是和微处理器(通常也和平台)相关。Intel中断处理代码在arch/i386/kernel/irq.c中,它的定义在incude/ asm-i386/irq.h。

Device Drivers(设备驱动程序):Linux核心源代码的大部分代码行在它的设备驱动程序中。Linux所有的设备驱动程序源代码都在drivers中,但是它们被进一步分类:

/block 块设备驱动程序。如果希望查看所有可能包含文件系统的设备是如何初始化的，可以看drivers/block/genhd.c中的device\_setup()。它不仅初始化硬盘,也初始化网络,因为安装nfs文件系统的时候需要网络。块设备包括基于IDE和SCSI设备。

/char 这里可以查看基于字符的设备,比如tty,串行口等。

/cdrom 所有的CDROM代码。在这里可以找到特殊的CD-Rom设备(比如 Soundblaster CDROM)。注意IDECD驱动程序是drivers/block中的ide-cd.c，而SCSICD驱动程序在drivers/scsi/scsi.c中。

/pciPCI伪驱动程序。这是一个观察PCI子系统如何被映射和初始化的好地方。AlphaAXPPCI整理代码也只能在arch/alpha/kernel/bios32.c中查看。

/scsi在这里不但可以找到所有的Linux支持的SCSI设备的驱动程序,也可以找到所有的SCSI代码。

/net 在这里可以找到网络设备驱动程序比如DECChip21040PCI以太网驱动程序在tulip.c中。

/sound 所有的声卡驱动程序的位置。

File Systems(文件系统):ext2文件系统的源程序都在fs/ext2子目录，数据结构的定义在include/linux/ext2\_fs.h、ext2\_fs\_i.h 和 ext2\_fs\_sb.h中。虚拟文件系统的数据结构在include/linux/fs.h中描述,代码在fs/\*中。Buffer cache 和 update核心守护进程都是用fs/buffer.c实现的。

Network(网络):网络代码放在net子目录,大部分的include文件在include/ net。BSD socket代码在net/socket.c,Ipv4 INET socket代码在net/ipv4/af\_in- et.c中。通用协议的支持代码(包括sk\_buff处理例程)在net/core中，TCP/IP网络代码在net/ipv4。网络设备驱动程序在drivers/net中。

Modules(模块):核心模块代码部分在核心，部分则在modules包中。核心代码全部在kernel/modules.c,数据结果和核心守护进程kerneld的消息则分别在 include/linux/module. h和 include/linux/kerneld.h中。也可在include/linux/ elf.h中查看一个ELF目标文件的结构。

### 2.1.2 编译Linux内核和模块

首先通过命令安装编译Linux内核所需要的包：

# sudo apt-get install libncurses5-dev make openssl libssl-dev bison flex

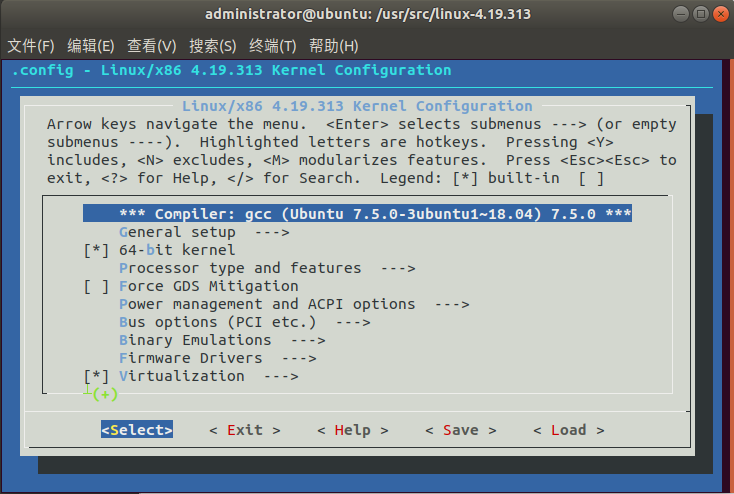
然后使用make mrproper清除目录下所有配置文件和先前生成核心时产生的.o文件：

# make mrproper

然后打开基于文本的选单式配置界面，创建config文件：

# make menuconfig

打开config窗口如下，使用默认配置即可，然后保存config：



随后开始编译Linux内核和模块，j4表示使用4线程进行编译，2> error.log表示将错误信息重定向到error.log文件下，便于后续查看：

# sudo make -j4 2> error.log

本人在编译Linux内核时还遇到了如下错误：

arch/x86/hyperv/hv\_apic.c: In function ‘\_\_send\_ipi\_one’:

arch/x86/hyperv/hv\_apic.c:197:1: warning: the frame size of 1032 bytes is larger than 1024 bytes [-Wframe-larger-than=]

}

^

arch/x86/hyperv/hv\_apic.c: In function ‘hv\_send\_ipi\_mask\_allbutself’:

arch/x86/hyperv/hv\_apic.c:222:1: warning: the frame size of 1032 bytes is larger than 1024 bytes [-Wframe-larger-than=]

}

^

arch/x86/xen/.tmp\_enlighten\_pv.o: warning: objtool: xen\_cpuid()+0x29: can't find jump dest instruction at .text+0xa0

make[1]: \*\*\* 没有规则可制作目标“debian/canonical-certs.pem”，由“certs/x509\_certificate\_list” 需求。 停止。

make: \*\*\* [certs] Error 2

make: \*\*\* 正在等待未完成的任务....

arch/x86/kernel/kvm.c: In function ‘kvm\_send\_ipi\_mask\_allbutself’:

arch/x86/kernel/kvm.c:519:1: warning: the frame size of 1032 bytes is larger than 1024 bytes [-Wframe-larger-than=]

}

^

表明 Makefile 正在寻找一个不存在的证书文件。使用如下命令禁用SYSTEM\_TRUSTED\_KEYS可解决该问题：

# scripts/config --disable SYSTEM\_TRUSTED\_KEYS

解决上述问题后，只需等待Linux内核编译完成即可，这个过程是很漫长的，本人在Ubuntu 18.04的虚拟机中编译需要半小时以上。

### 2.1.3 安装Linux内核和模块

通过上述步骤我们已经编译好了Linux内核，下面需要进行内核和模块的安装：

# sudo make modules\_install install

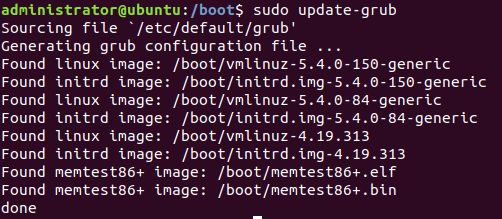
在得到所需版本内核后，我们还需修改/boot下的两个链接System.map和vmlinuz，使其指向新内核的文件：

#cd /boot;rm -f System.map vmlinuz

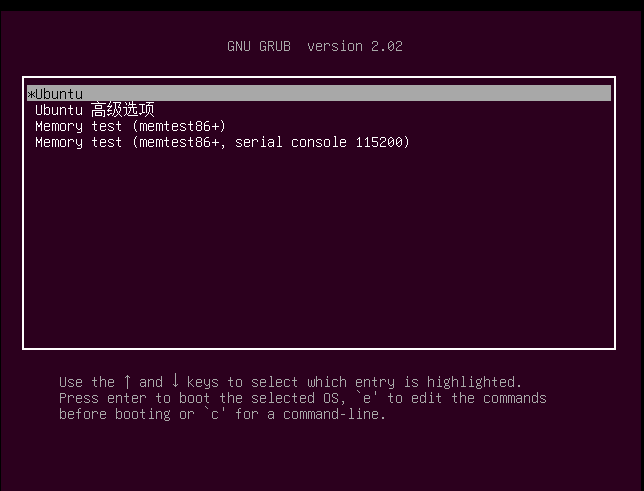
#ln -s vmlinuz-2.4.18 vmlinuz

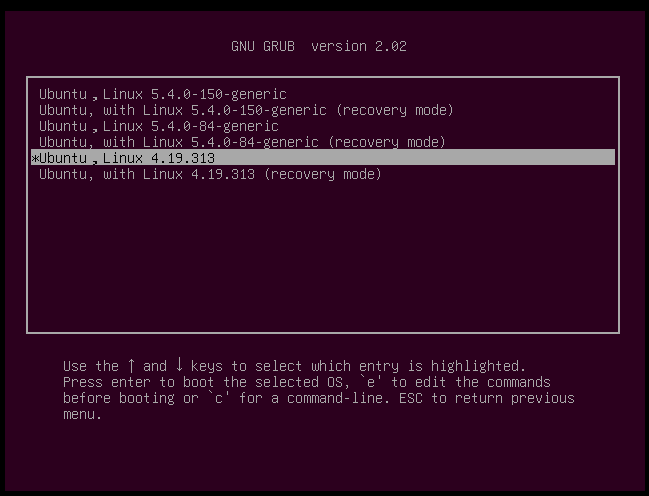
#ln -s System.map-2.4.18 System.map

使用sudo update-grub命令更新grub，将新的内核添加进grub引导程序：

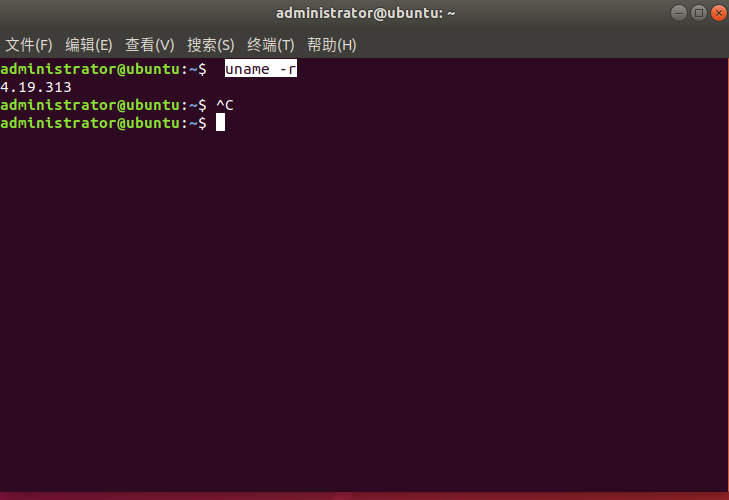


重启系统，在启动时长按shift进入grub界面，选择“Ubuntu高级选项”并选择之前新安装的内核：





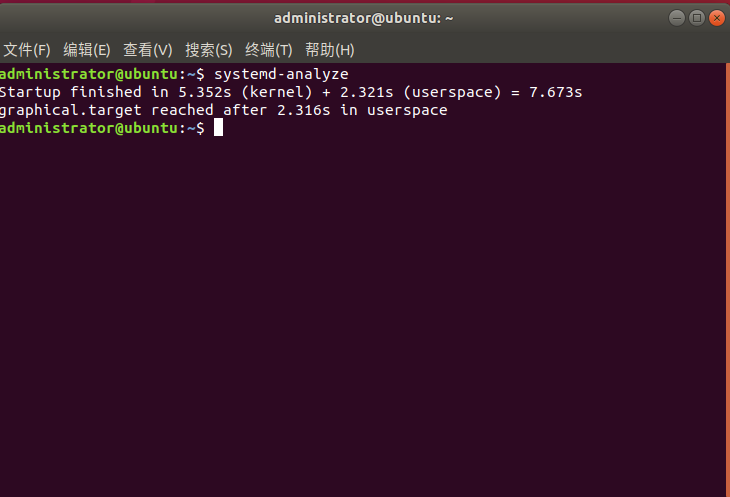
使用命令uname -r查看Linux内核版本，可以看到，系统已经使用了新安装的内核版本4.19.313：



## 2.2 分析系统启动时间

### 2.2.1测定Linux系统启动时间

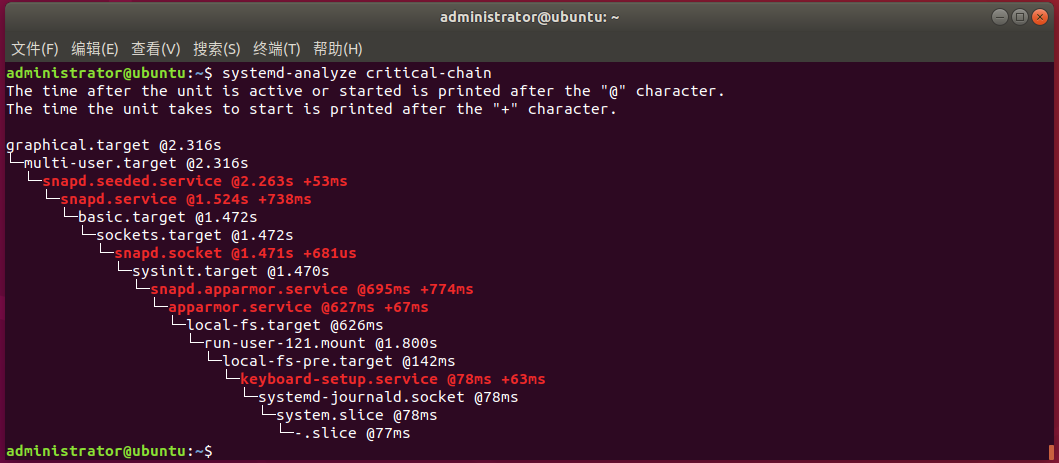
systemd-analyze 可以显示从引导加载程序（boot loader）到用户空间（user space）以及最终完全启动的总时间。可以使用systemd-analyze命令来测定系统的启动时间：



可以看到，初始系统启动时间为7.673s。

### 2.2.2分析系统启动时间

使用systemd-analyze critical-chain显示启动过程中服务的依赖关系及其启动时间，借此来分析Linux启动的过程及其占用时间：



以下是分析系统启动过程的每个阶段及其耗时：

graphical.target @2.316s:系统图形界面目标在2.316秒后启动，这是最终用户可见的桌面环境。

multi-user.target @2.316s:多用户目标在2.316秒后启动，这是图形目标的依赖项，提供了多用户模式支持。

snapd.seeded.service @2.263s +53ms:snapd.seeded.service 在2.263秒后启动，并在53毫秒内启动完成。此服务确保snapd种子配置已经应用。

snapd.service @1.524s +738ms:snapd.service 在1.524秒后启动，并在738毫秒内启动完成。这是管理snap包的主要服务，启动时间相对较长。

basic.target @1.472s:基础目标在1.472秒后启动，表示系统的基本服务已经启动完成。

sockets.target @1.472s:套接字目标在1.472秒后启动，表示所有需要的套接字已被创建。

snapd.socket @1.471s +681us:snapd.socket 在1.471秒后启动，并在681微秒内完成。这是snapd服务的套接字。

sysinit.target @1.470s:系统初始化目标在1.470秒后启动，表示系统初始化过程已经完成。

snapd.apparmor.service @695ms +774ms:snapd.apparmor.service 在695毫秒后启动，并在774毫秒内启动完成。这是snapd服务的安全策略管理服务，启动时间较长。

apparmor.service @627ms +67ms:apparmor.service 在627毫秒后启动，并在67毫秒内启动完成。它负责加载和管理AppArmor安全配置。

local-fs.target @626ms:本地文件系统目标在626毫秒后启动，表示本地文件系统已经挂载完成。

run-user-121.mount @1.800s:挂载用户121的运行目录在1.800秒后完成，这一过程是在本地文件系统目标之后完成的。

local-fs-pre.target @142ms:本地文件系统预目标在142毫秒后启动，这是在本地文件系统挂载前的准备工作。

keyboard-setup.service @78ms +63ms:键盘设置服务在78毫秒后启动，并在63毫秒内启动完成，负责设置键盘布局。

systemd-journald.socket @78ms:systemd-journald.socket 在78毫秒后启动，这是系统日志守护进程的套接字。

system.slice @78ms:系统切片在78毫秒后启动，表示系统服务的分组。

-.slice @77ms:根切片在77毫秒后启动，表示系统启动的最基本单位。

# **第三章 实验总结**

在本次实验中，我下载、编译并安装了Linux内核，替换了原始Linux内核，并使用命令测定并分析了Linux系统的启动时间。通过这次实验，我不仅加深了对Linux系统启动的理解，也掌握了安装Linux系统的实用技能，同时加深了对Linux命令的熟悉程度。这些技能对于今后在软件开发和维护的过程中将非常有用。