**操作系统课程实践实验报告**

实验题目：添加系统调用并构建内核

姓 名：

学 号：

组 号：

专 业：

班 级：

老师姓名：

日 期：2023年10月29日星期日

目 录

[一 题目介绍 1](#_Toc515881940)

[二 实验思路 2](#_Toc515881941)

[三 遇到问题及解决方法 1](#_Toc515881942)1

[四 核心代码及实验结果展示 11](#_Toc515881943)

[五 个人实验改进与总结 18](#_Toc515881944)

[5.1 个人实验改进 18](#_Toc515881945)

[5.2 个人实验总结 18](#_Toc515881946)

[六 参考文献 19](#_Toc515881947)

# 一 题目介绍

## 实验目的

Linux 是开源操作系统，用户可以根据自身系统需要裁剪、修改内核，定制出功能更加 合适、运行效率更高的系统，因此，编译linux内核是进行内核开发的必要基本功。在系统中根据需要添加新的系统调用是修改内核的一种常用手段，通过本次实验，应理解 linux 系统处理系统调用的流程以及增加系统调用的方法。

## 实验内容

（1）添加一个系统调用，实现对指定进程的 nice 值的修改或读取功能，并返回进程的 nice 值及优先级 prio。建议调用原型为： int mysetnice(pid\_t pid, int flag, int nicevalue, void \_\_user \* prio, void \_\_user \* nice); 参数含义： pid：进程 ID。 flag：若值为 0，表示读取 nice 值；若值为 1，表示修改 nice 值。 Prio、nice：进程当前优先级及 nice 值。 返回值：系统调用成功时返回 0，失败时返回错误码 EFAULT。

（2）写一个简单的应用程序测试（1）中添加的系统调用。

（3）若程序中调用了 linux 的内核函数，要求深入阅读相关函数源码。

## 试验环境

用busybox和linux内核构建一个最小的执行环境, 并用qemu来启动内核进行验证

linux-5.15.137

busybox-1.34.1

QEMU emulator version 8.1.2

什么是busybox

busybox是linux下的一套精简工具集和内核一起构成了最小的执行环境

# 二 实验思路

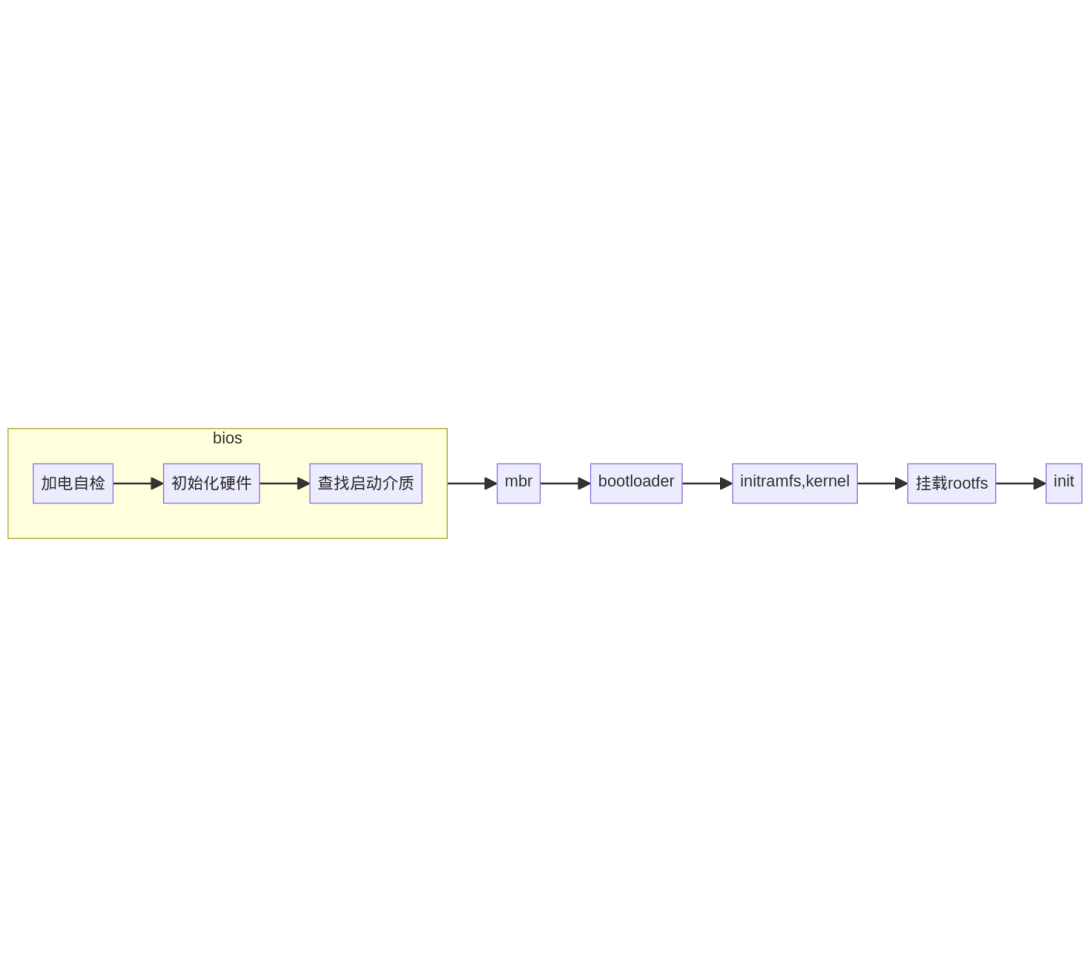


图 1linux启动流程

其中bootloader和之前的部分qemu已经帮我们完成了， 所以我们只需要完成后面的部分，另外我们并不是实现一个完整的操作系统环境，因此我们不必挂载真正的根文件系统，直接拿initramfs作为最终的根文件系统就行了

## 代码树



图 2代码树

下载内核源码



图 3脚本

修改系统调用



图 4添加操作系统条目



图 5定义新系统调用

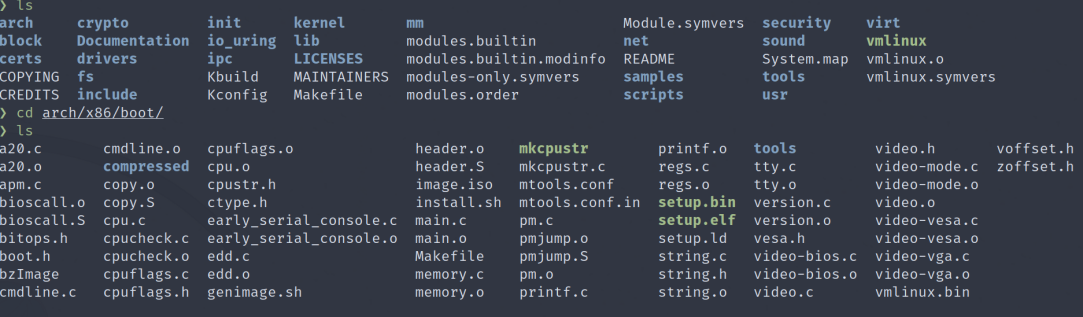


图 6用脚本修改源码

构建内核



图 7脚本



vmlinux和bzImage就是我们希望得到的构建产物(内核镜像和gzip压缩后的内核镜像)

下载busybox源码



图 8脚本

构建busybox

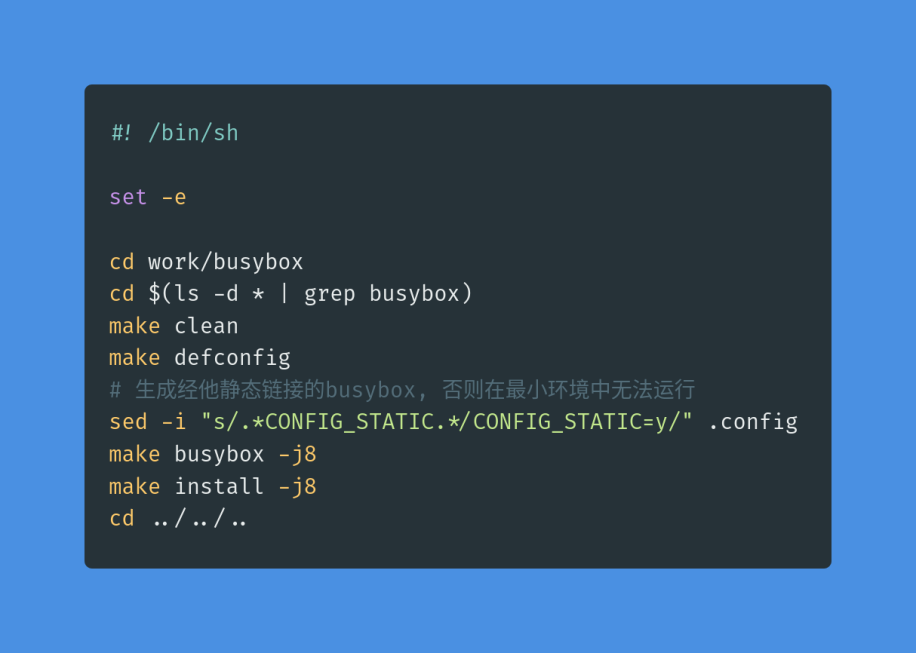


图 9脚本

必须采用静态编译, 在我们的最小环境下glibc是缺失的

构建完成后会在源码根目录下生成一个\_install目录, 我们将以该目录为基础制作根文件系统

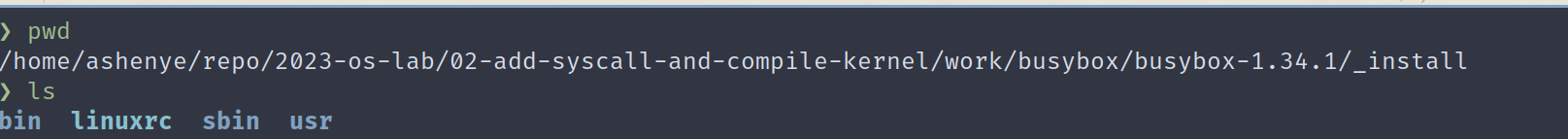


图 10生成的目录

构建用户态程序

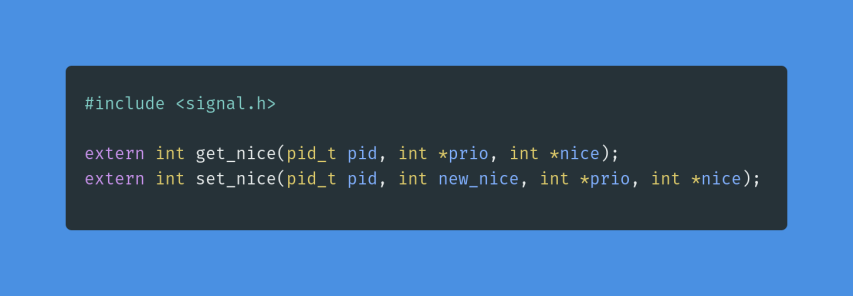
我们不希望用户态程序直接发起系统调用,而是把系统调用封装成库函数，用户态程序并不直接调用系统调用，而是调用系统调用间接地调用库函数，这样大大降低了出错的可能性

图 11头文件

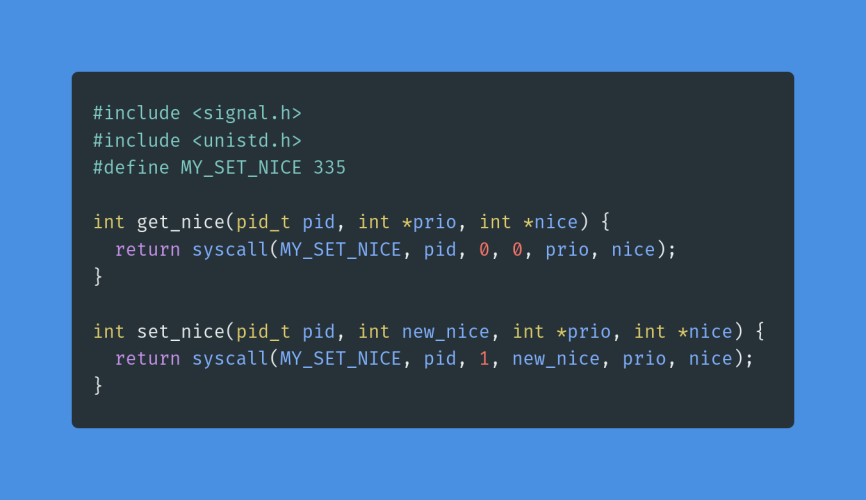


图 12定义



图 13用户态程序

这个程序的功能是根据传进来的参数来展示或者修改优先级

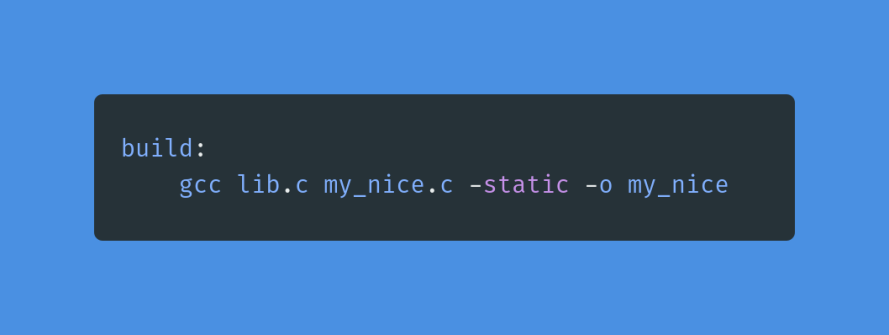


图 14构建脚本

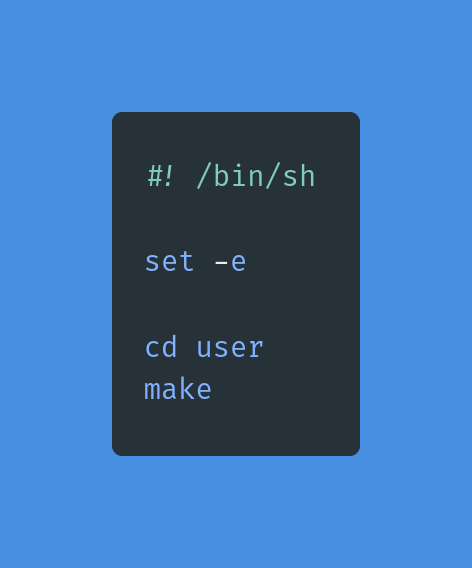


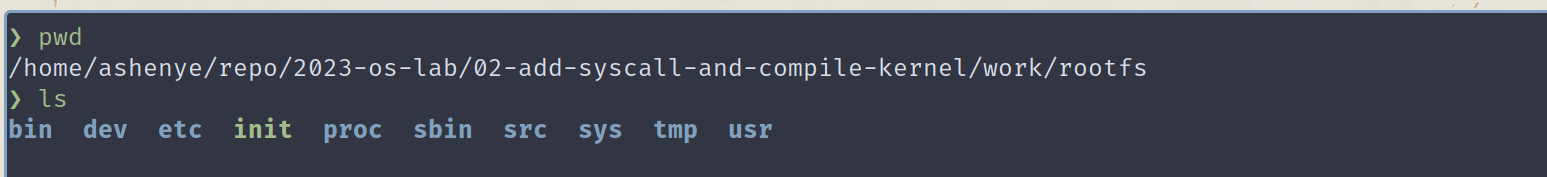
图 15构建脚本

构建根文件系统



图 16构造根文件系统

执行完后就有一个完整的根文件系统了



把根文件系统打包



图 17脚本

执行完后生成一个根文件系统镜像

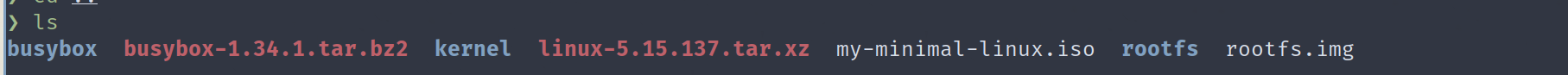
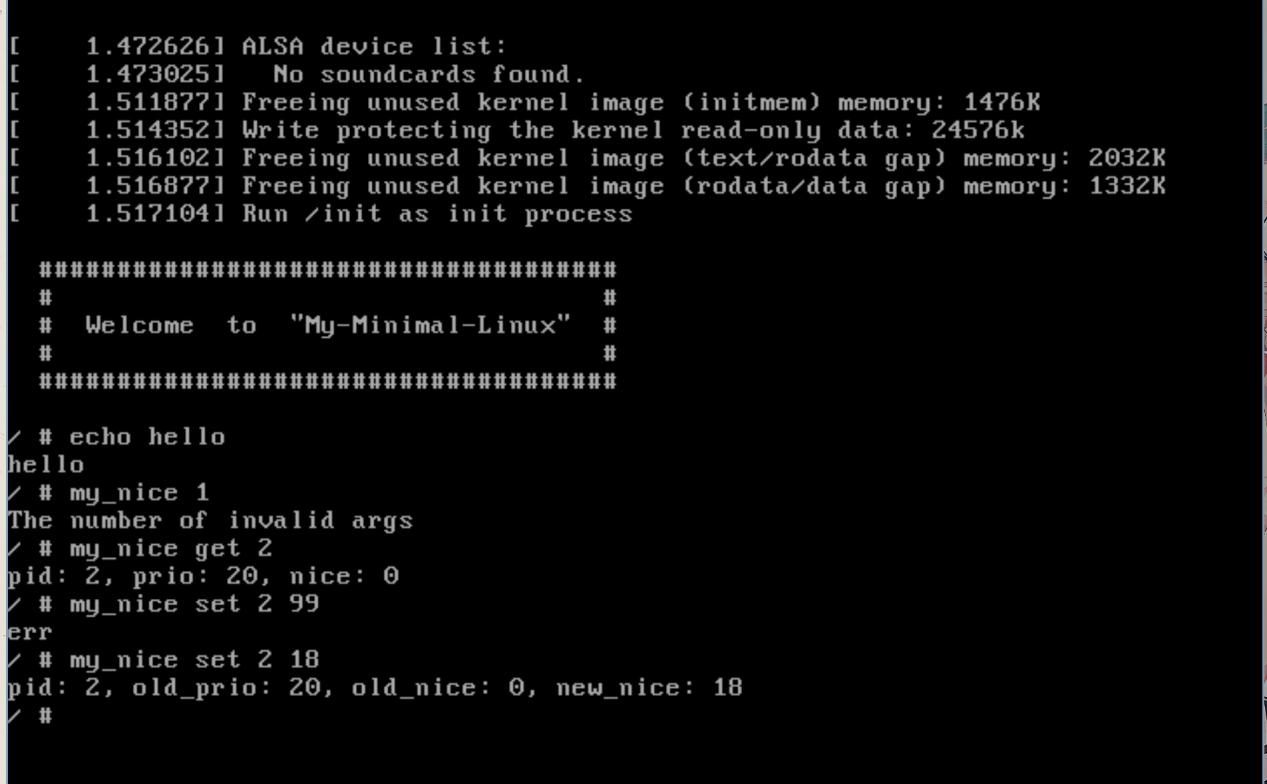


图 18rootfs.img

qemu启动





# 三 遇到问题及解决方法

1. 找不到符号`sys\_my\_set\_nice`

ld: arch/x86/entry/syscall\_64.o:(.rodata+0xa68): undefined reference to `sys\_my\_set\_nice`

make: \*\*\* [Makefile:1049: vmlinux] Error 1

系统调用命名不规范, 定义处是不需要加`sys\_`前缀的

1. 编译busybox时`/usr/bin/ld: cannot find -lcrypt: No such file or directory`

我用的linux发行版为了裁剪体积或者强制动态链接的未来证明安全性把libcrypt.a移除掉了用其他发行版的软件仓库的libcrypt.a就行了

1. 静态编译busybox失败

# Static linking against glibc, can't use --gc-sections

# Trying libraries: crypt m

# Failed: -Wl,--start-group -lcrypt -lm -Wl,--end-group

glibc好像和静态链接不太对付, 换用musl作为c语言库

# 四 核心代码及实验结果展示



图 19系统调用代码

find\_get\_pid分析

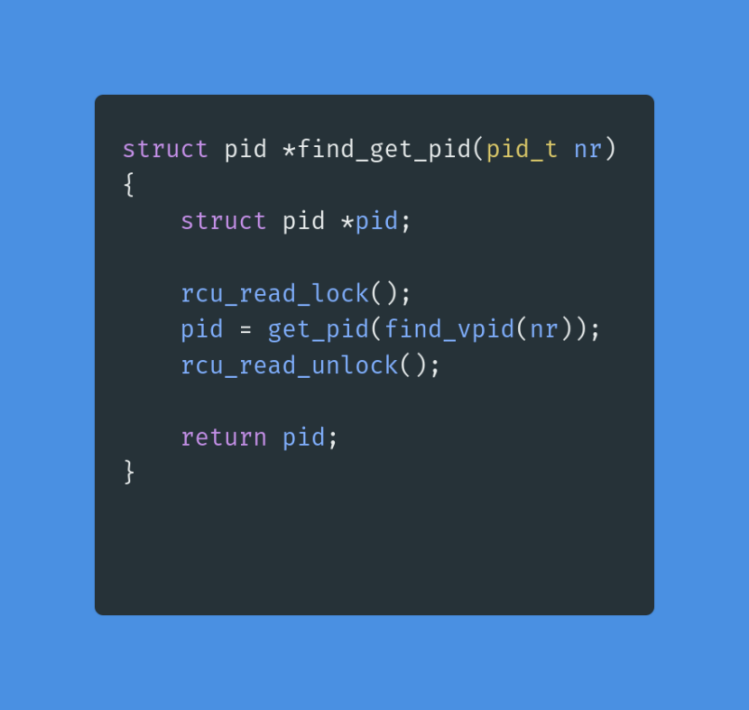


图 20通过pid得到pid结构体指针（加了rcu读锁保护）

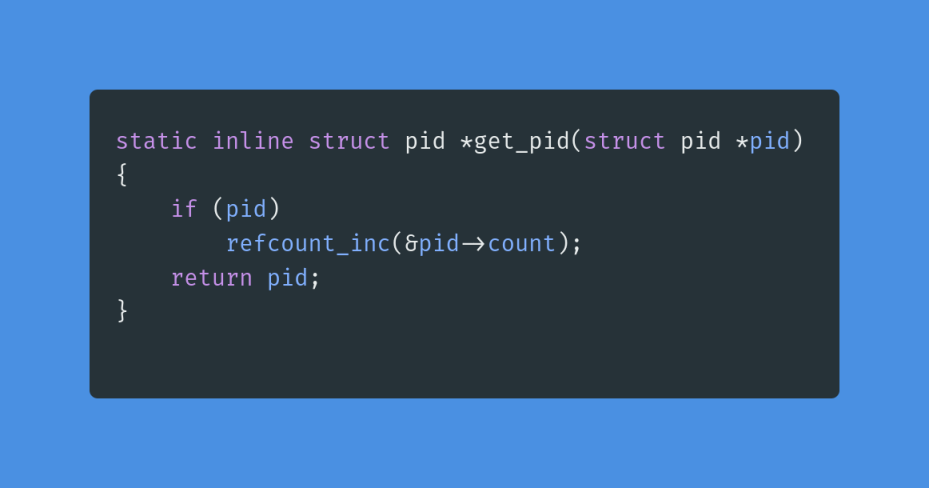


图 21pid描述符内部的原子引用计数加1

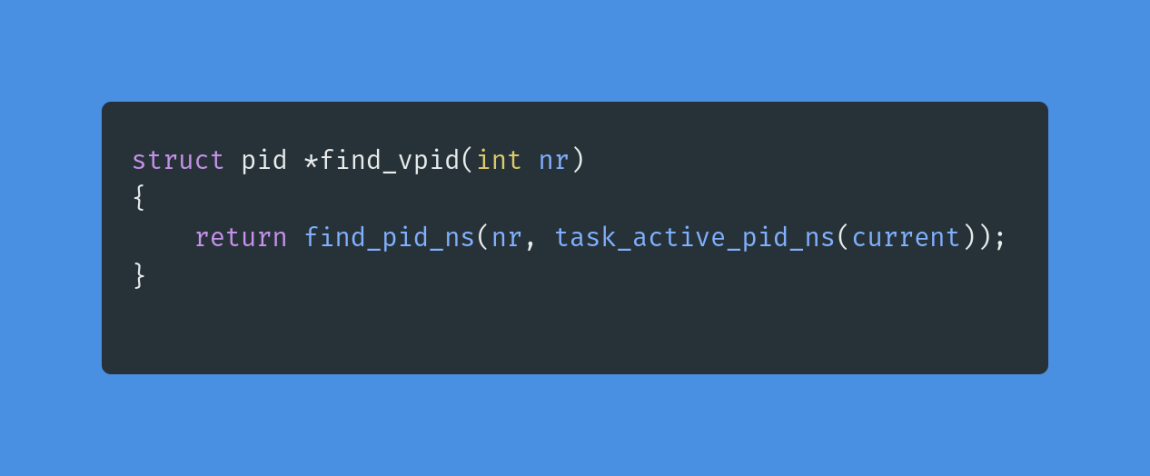


图 22在当前的命名空间下查找pid

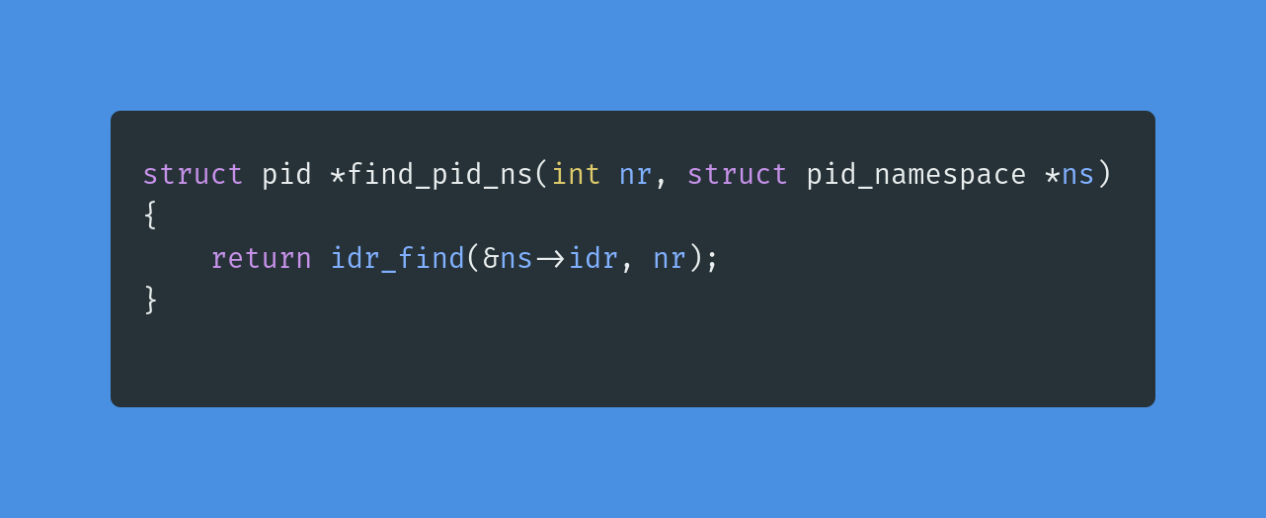


图 23得到给定id的指针

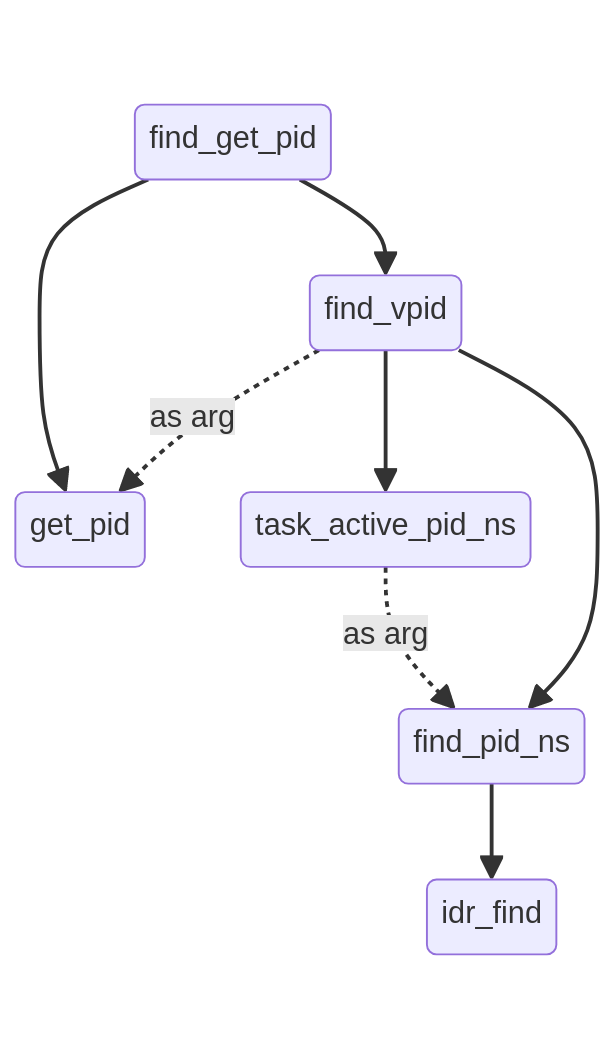


图 24函数引用图

find\_get\_pid整个函数是被rcu读锁保护着的。find\_get\_pid先去调用find\_vpid, find\_vpid再去调用find\_pid\_ns即在当前current的命名空间下查找pid, find\_get\_pid再把返回结果传给get\_pid这个函数负责把pid内部的引用计数加1，然后再把pid返回出来。最终把结果返回

整体流程概述

首先判断传参是否合法, 不合法直接抛出错误 然后通过pid得到要找的task的地址。然后把prio和nice取出来。如果flag为1则用新值更新nice和prio最后把取到的prio和nice拷贝到用户态空间返回成功

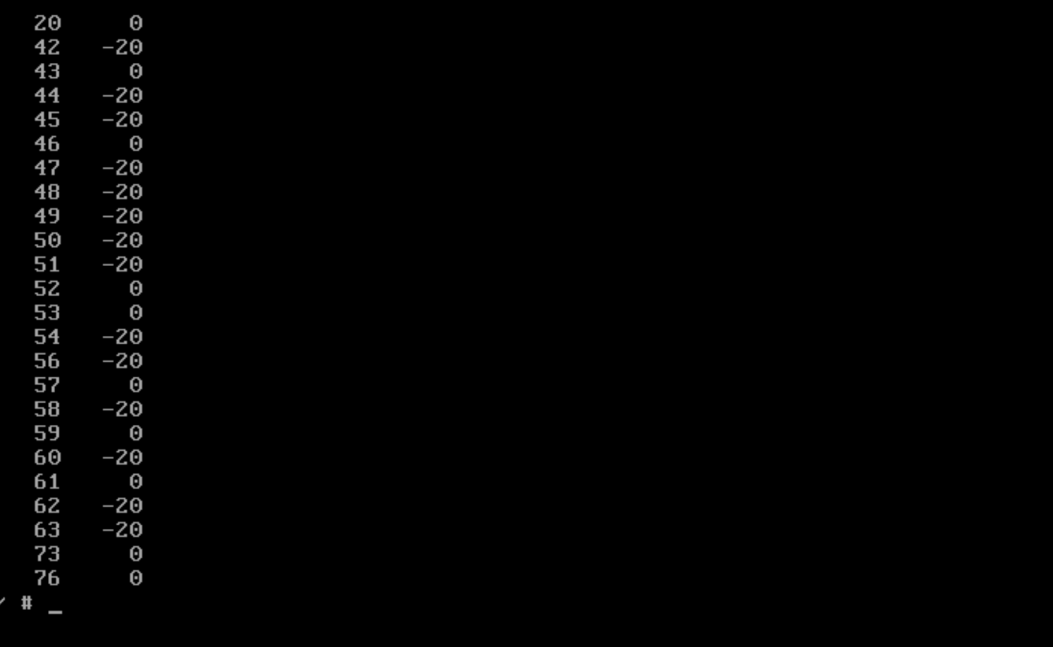


图 25进程63的nice值为-20

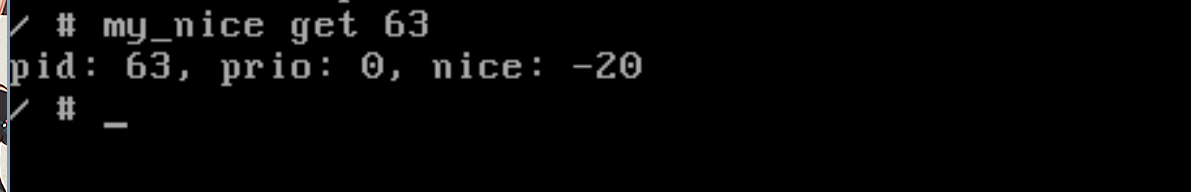


图 26用my\_nice读取结果一致

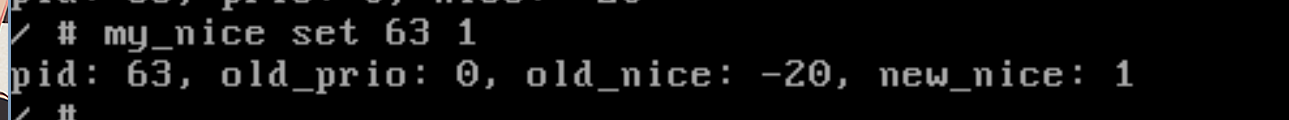


图 27用my\_nice修改nice值为1

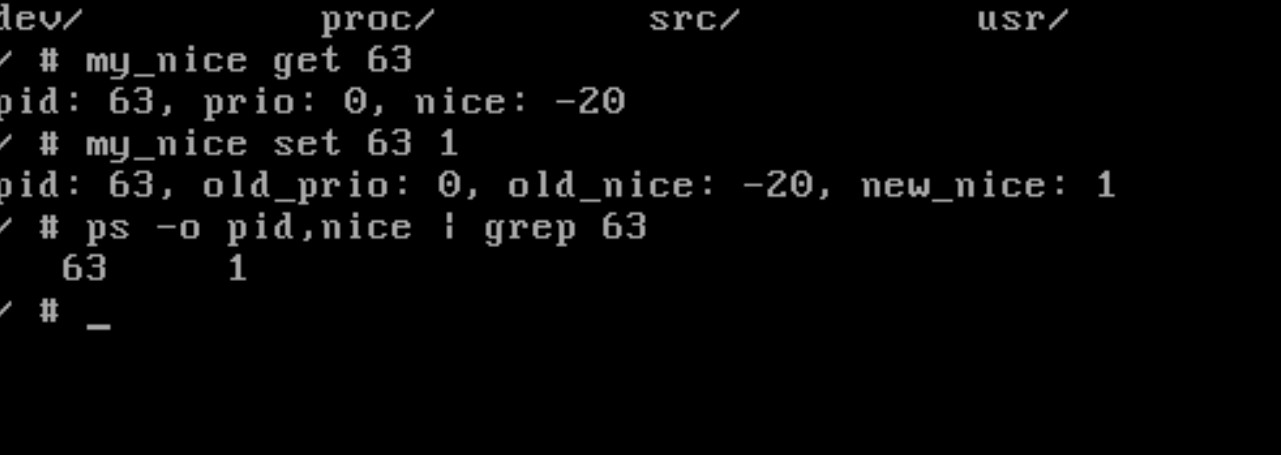


图 28用ps命令可以看到修改成功

# 五 个人实验改进与总结

## 5.1 个人实验改进

## 大部分同学都是用vmware来实验的， 最大的不同是我用的是linux+busybox+qemu三者的组合

这种改进实验步骤的方式可以带来以下优势：

1. 更接近实际环境：使用Linux+BusyBox+QEMU的组合更接近于真实的操作系统开发环境。在实际开发中，通常会与原生Linux系统打交道，而不是虚拟机。这种方式可以帮助你更好地理解实际工作中的情境和挑战。

2. 更大的自主性：通过使用Linux+BusyBox+QEMU，需要自己配置和搭建实验环境。这鼓励了自主学习和问题解决能力的培养，因为你需要更深入地了解每个组件如何工作以及它们如何协同工作。

3. 更全面的学习：使用VMware可能隐藏了一些操作系统底层的细节，而Linux+BusyBox+QEMU的组合能让你更全面地了解操作系统的内部机制，包括内核编译、文件系统构建和系统调用的添加等。

4. 更高的可定制性：Linux+BusyBox+QEMU组合允许你自定义实验环境，以适应不同的实验需求。可以精简内核、添加自定义组件，或者配置QEMU以模拟不同的硬件环境。这种可定制性有助于更好地满足特定实验的要求。

5. 实际调试经验：通过使用QEMU进行实验，可以获得更多实际的调试经验，因为QEMU提供了丰富的调试工具和选项。这将使能够更好地理解操作系统中的错误和问题，并学会如何解决它们。

总的来说，使用Linux+BusyBox+QEMU的组合改进实验步骤有助于更深入地学习操作系统原理和开发，培养自主学习和解决问题的技能，以及为未来的操作系统开发和系统编程提供更实际的经验。这种方式有助于更全面地掌握操作系统的核心概念和技术。

## 5.2 个人实验总结

当我学习如何添加系统调用以及构建最小的操作系统执行环境时，我收获了许多宝贵的经验和技能。这次的实验让我更深入地理解了操作系统的内部工作原理，并提供了有关系统调用和操作系统构建的宝贵见解。以下是我在这次实验中的心得体会：

首先，学习如何添加系统调用让我更好地理解了操作系统与用户程序之间的交互方式。系统调用是用户程序与内核之间的桥梁，允许用户程序请求操作系统的服务。通过学习如何添加自定义系统调用，我深刻体会到了系统调用表、系统调用号和参数传递等概念的重要性。这不仅提高了我的编程技能，还让我了解了操作系统内核的核心功能。

其次，使用BusyBox和Linux内核构建最小的操作系统执行环境是一项令人兴奋的任务。这个过程涉及到配置内核选项、构建文件系统、编译工具链等多个步骤。通过亲自动手完成这些任务，我深刻认识到了操作系统的层次结构和内核的关键作用。我学会了如何精简内核和文件系统，以创建适合特定需求的最小化操作系统。

最后，使用QEMU进行调试验证是实验的重要一部分。QEMU是一个功能强大的虚拟化工具，它允许我们在模拟的环境中测试和调试操作系统。通过与QEMU的互动，我学会了如何跟踪系统调用的执行、调试内核代码以及捕捉和分析系统日志。这些技能对于操作系统开发和故障排除非常有帮助。

总的来说，这次实验为我提供了深入了解操作系统的机会，让我对系统调用、内核构建和调试工具有了更全面的认识。这些知识和技能将对我的计算机科学和工程学习道路产生积极影响，并为我未来的职业生涯打下坚实的基础。我期待着进一步深入研究操作系统和探索更多相关领域的机会。

# 六 参考文献

[the\_dao\_of\_minimal\_linux\_live.txt](https://ivandavidov.github.io/minimal/the_dao_of_minimal_linux_live.txt)

[Minimal Linux Live](https://github.com/ivandavidov/minimal)

<https://zhuanlan.zhihu.com/p/469972204>

<https://zhuanlan.zhihu.com/p/424240082>