操作系统课程设计实验报告

*赵宇阳*

*22009200439*

1 实验1 Kernel API

实验目的

通过本实验的学习，掌握信创操作系统内核定制中所常用的内核数据结构和函数，具体包括：

i.内核链表；

ii.内核内存的分配释放；

iii.内核线程；

iv.内核锁

实验内容

1. 设计一个内核模块，并在此内核模块中创建一个内核链表以及两个内核线程。

线程I需要遍历进程链表并将各个进程的pid、进程名加入到内核链表中。

线程2中需不断从内核链表中取出节点并打印该节点的元素。

1. 在卸载模块时停止内核线程并释放资源。

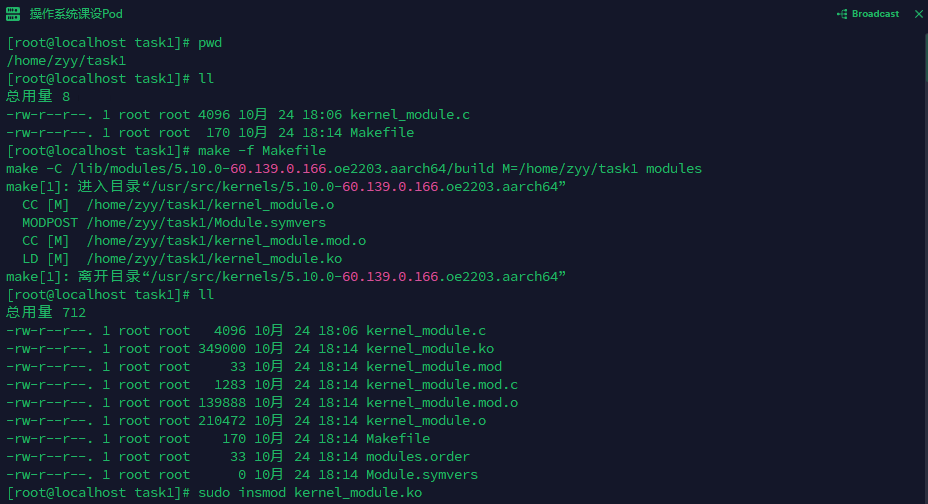
实验步骤

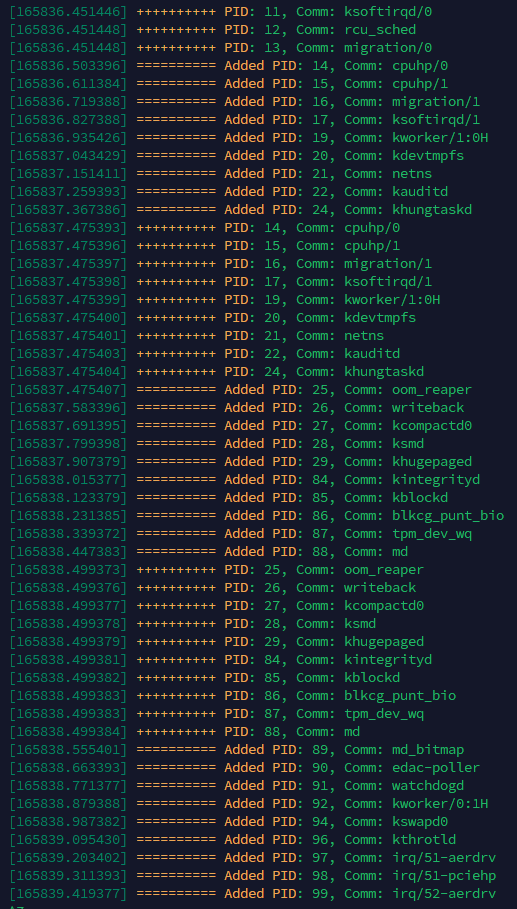
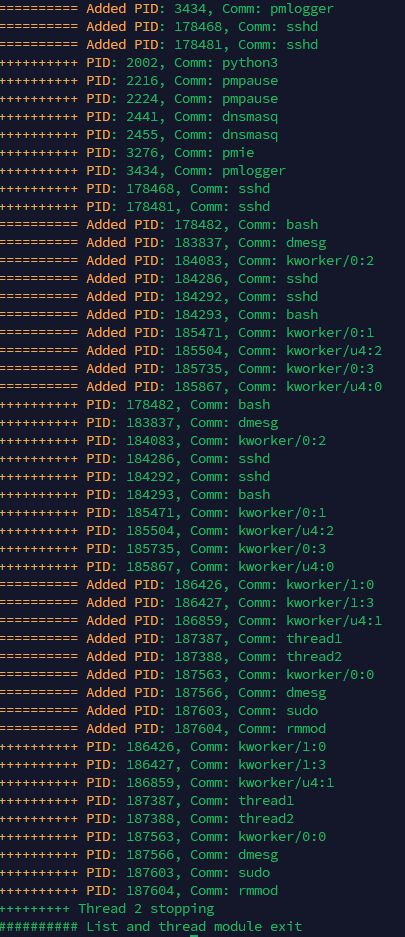
1. 根据给出的kernel\_module.c框架代码，进行补全

实现逻辑：线程1遍历所有进程将每个进程的PID和命名名称添加到链表中；线程2从链表中取出并删除节点、释放内存；模块初始化时初始化链表和自旋锁创建并启动两个内核线程；模块卸载时停止并清理两个内核线程，释放链表中的所有节点

1. 根据给出的Makefile进行编辑改写，实现对当前文件的编译链接支持
2. 使用远程主机连接管理工具Terminus连接Pod，再连接到虚拟机，使用vim将本地的kernel\_module.c文件和Makefile中的内容粘贴到工作目录中
3. 编译链接产生kerner\_module.ko模块文件
4. 装载模块，并使用dmesg命令查看内核输入日志
5. 确定运行没有问题之后卸载模块

实验结果





关键问题

1. kmalloc和kfree用于内核空间，在内核模块或驱动程序中使用，分配释放内核内存；C语言中的malloc和free用于用户空间内存，分配释放用户空间内存
2. 使用for\_each\_process宏来遍历所有进程并获取每个进程的struct task\_struct指针
3. 在两个线程中都使用spin\_lock宏来进行锁的获取，使用spin\_unlock宏来进行锁的释放，以此来实现同步互斥
4. 内核线程在内核态执行，直接被内核抢占和调度，具有更高权限，可以直接执行系统调用，上下文切换涉及内核态的切换，开销较大；用户空间线程在用户态执行，权限较低，不能直接访问内核数据结构和硬件资源，调度粒度较粗，上下文切换开销较小
5. 内核链表已经经过应用测试可用性和安全性高，且针对内核环境进行了优化，包括内存分配、数据结构布局等，性能更好

实验2 deferred work

实验目的

通过本实验的学习，掌握信创操作系统内核定制中所常用的workqueue的使用，理解其与kernel thread 的区别

实验内容

设计并实现一个内核模块，该模块旨在通过work queue 和kernel thread两种不同的机制来完成延迟工作（deferred work），并对比分析

这两种实现方式的异同点。

分别采用workqueue 和kernel thread两种方式调用10个函数（函数内部打印学号后3位依次加1的方式区分。例如函数I中打印315，函数2中打印316，以此类推），观察并记录 work queue 与kernel thread 在执

行函数时的顺序差异。请注意，每个函数应当对应一个独立的kernel thread，即10个函数需由lo个不同的kernel thread 分别执行。

探究 work queue 中的 delayed\_work功能，要求在模块加载成功的5秒后打印一条预设的信息，以验证delayed\_work的延迟执行效果。

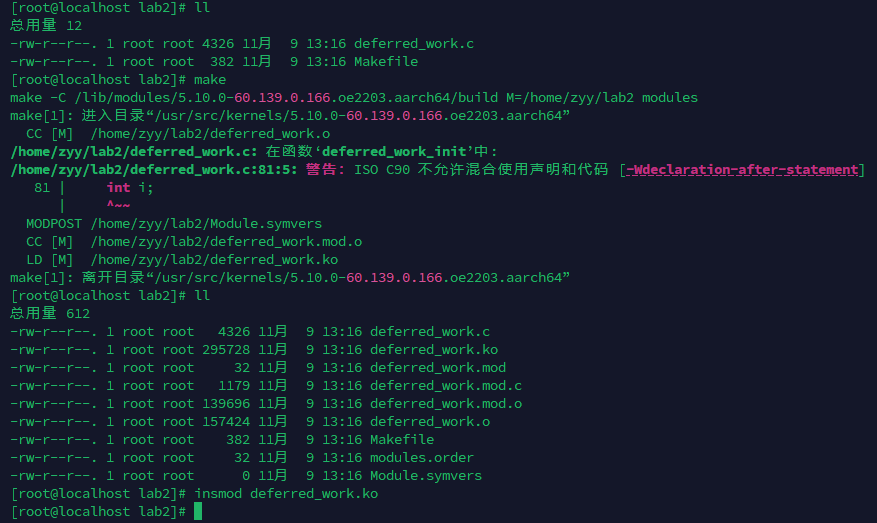
实验步骤

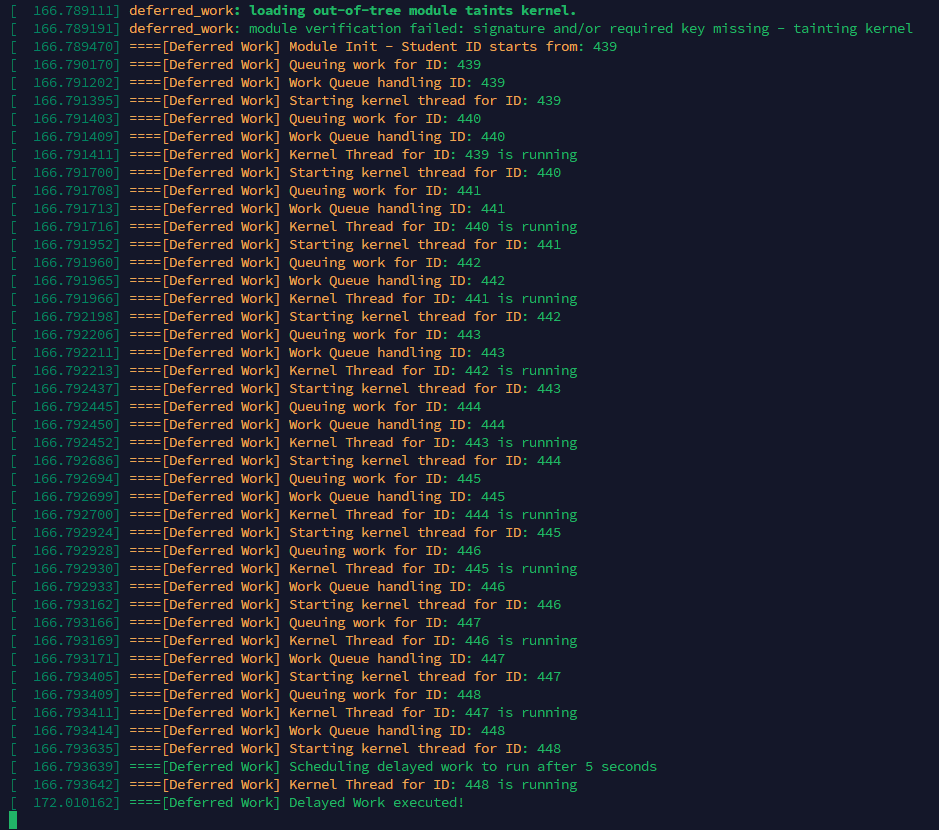
1. 根据给出的deferred\_work.c框架代码，进行补全

实现逻辑：模块初始后时打印初始化信息：输出模块初始化的信息，包括起始的学生ID；创建工作队列；初始化并排队工作项，创建内核线程；初始化并调度延迟工作；当模块退出时停止所有内核线程并销毁工作队列

1. 根据给出的Makefile进行编辑改写，实现对当前文件的编译链接支持
2. 使用远程主机连接管理工具Terminus连接Pod，再连接到虚拟机，使用vim将本地的deferred\_work.c文件和Makefile中的内容粘贴到工作目录中
3. 编译链接产生deferred\_work.ko模块文件
4. 装载模块，并使用dmesg命令查看内核输入日志
5. 确定运行没有问题之后卸载模块

实验结果





关键问题

workqueue工作队列可以理解为一种异步任务执行机制，允许将任务放入队列中由内核管理的线程池中的工作线程来执行这些任务，支持多线程；kernel thread内核线程是运行在内核态的线程，由内核调度器管理，可以执行长时间运行的任务或阻塞操作，每个线程上下文都有自己的上下文，可以独立执行任务，复杂度和使用更为复杂

实验3 Edu驱动

实验目的

通过本实验的学习，掌握信创操作系统内核定制中所常见 PCI设备驱动适配技术

实验内容

补全edu设备驱动的框架代码

1. edu\_driver\_probe·

为 edu\_dev\_info 实例分配内存；

将BAR的总线地址映射到系统内存的虚拟地址

1. edu\_driver\_remove

从设备结构体 dev 中提取edu\_dev\_info实例；

补全iounmap函数的调用的参数；

释放 edu\_dev\_info 实例；

1. kthread\_handler

将用户传入的变量交给edv设备进行阶乘计算，并读出结果，注意加锁。结果放入user\_data 中的 data数据成员中时，需要确保读写原子性

1. edu\_dev\_open

完成filp 与用户进程上下文信息的绑定操作

1. edu\_dev\_release

释放edu\_dev\_open 中分配的内存

1. edu\_dev\_unlocked\_ioctl

用户通过ioctl传入要计算阶乘的数值，并读取最后阶乘的结果。计算阶乘使用内核线程，线程代码放在kthread\_handler中；

该调用需接收一个整型参数。当驱动程序接收到用户的ioctl调用后，需创建一个内核线程。在该内核线程中，利用edu设备的阶乘功能对传入的整型参数进行计算，并将计算结果存储于驱动程序中，以便用户进程后续获取。

驱动程序需具备识别不同进程调用的能力，确保将计算结果正确返回给对应的调用进程。

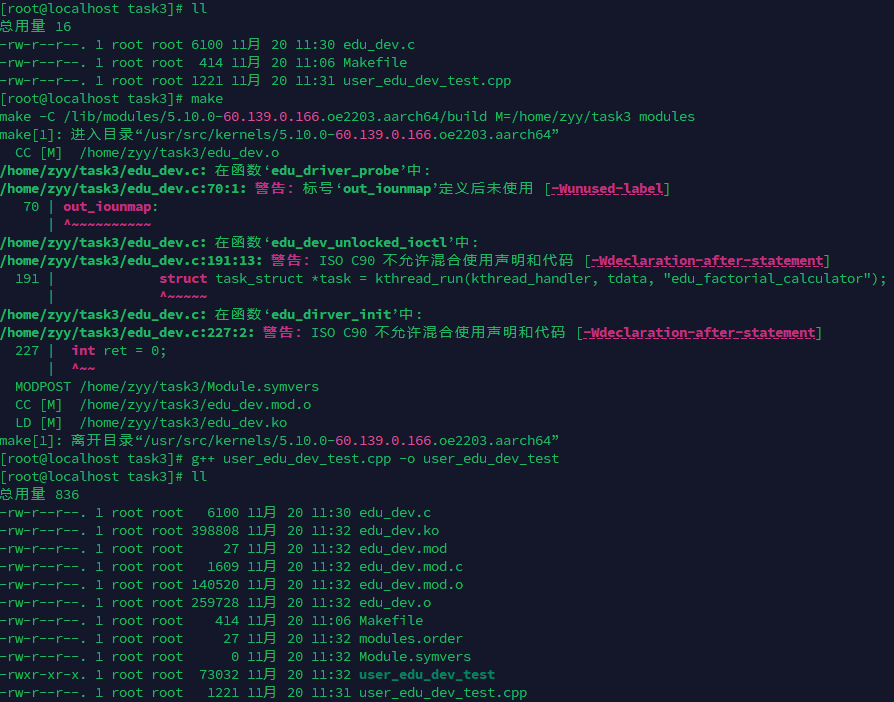
编写C语言程序，通过调用edu驱动的ioctl接口进行操作

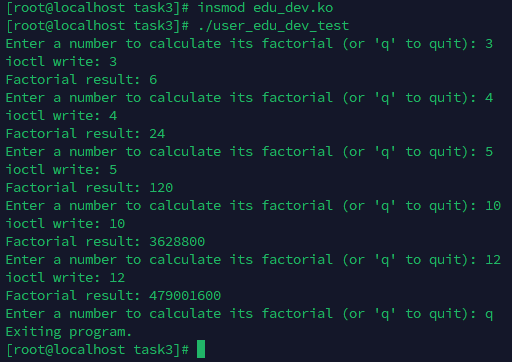
首先，设置参数cmd值为0，输入待计算的数值。等待一定时间后，将参数cmd 值更改为l，再次调用ioctl 接口，以获取设备计算完成的结果。

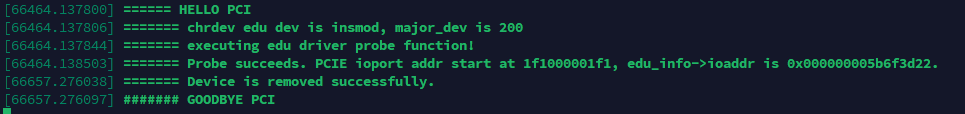
实验步骤

1. 根据给出的框架代码，进行补全
2. 定义设备信息结构体；
3. 定义设备ID表
4. 实现设备探测函数：启动设备、分配内存保存设备信息、请求PCI资源区域、映射I/O地址、设置设备私有数据指针、打印信息
5. 实验设备移除函数：取消I/O地址映射、释放PCI资源、释放内存、禁用PCI设备、打印信息
6. 定义字符设备操作
7. 实现字符设备操作函数：分配并初始化以及释放用户数据结构、处理设备ioctl命令—计算阶乘并获取计算结果
8. 驱动程序初始化：注册字符设备、注册PCI设备驱动、初始化自旋锁
9. 驱动程序卸载：注销字符设备和PCI设备驱动、打印卸载信息
10. 实现用户交互程序，循环写入一个数并计算阶乘打印结果
11. 根据给出的Makefile进行编辑改写，实现对当前文件的编译链接支持
12. 使用远程主机连接管理工具Terminus连接Pod，再连接到虚拟机，使用vim将本地文件的内容粘贴到工作目录中
13. 编译链接产生edu\_dev.ko模块文件和user\_edu\_dev\_test可执行文件
14. 装载模块，并使用dmesg命令查看内核输入日志
15. 运行可执行文件，进行交互，输入数据并查看计算结果
16. 确定运行没有问题之后卸载模块

实验结果







关键问题

ii. 先请求和映射PCI设备的I/O资源区域，然后通过映射后的指针访问具体的寄存器地址。在实验代码中，在edu\_driver\_probe函数中，调用pci\_request\_regions函数来请求PCI设备的I/O资源区域，使用pci\_ioremap\_bar函数将PCI设备的第一个BAR（Base Address Register）映射到内核虚拟地址空间，一旦I/O内存空间被映射，就可以通过这个指针来访问设备的寄存器

iii.

好处：

避免阻塞主进程。计算阶乘是一个耗时的操作。如果直接在ioctl调用中进行计算，可能会导致用户空间的应用程序长时间等待，影响用户体验。通过创建一个内核线程来处理计算任务，可以避免阻塞主进程，使得用户空间的应用程序能够继续执行其他操作。

简化代码逻辑。将计算任务分离到独立的线程中，可以使ioctl处理函数更加简洁，专注于命令的分发和参数的传递。复杂的计算逻辑则由专门的线程来处理。

带来的问题：

线程管理开销。创建和管理线程本身需要一定的系统资源和时间频繁地创建和销毁线程可能会导致额外的开销。

同步问题。代码中，使用了自旋锁spinlock\_t lock来保护共享数据atomic64\_t data。虽然这解决了部分同步问题，但在高负载情况下，自旋锁可能导致CPU占用率升高。

调试难度增加。

实验4 内核裁剪

实验目的

通过本实验的学习，掌握信创操作系统内核裁剪、根文件系统定制以及内核调参技术。

实验内容

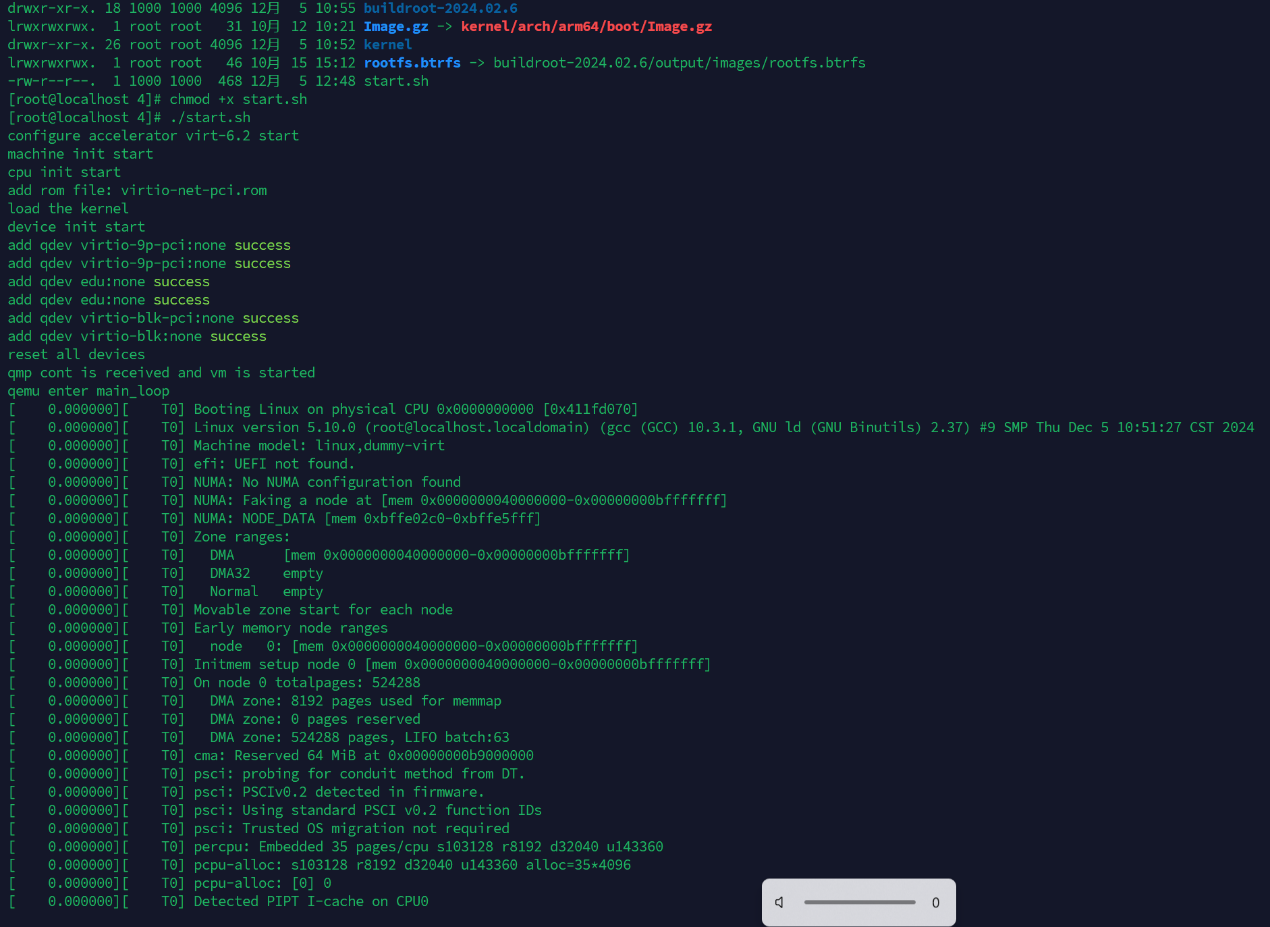
在本次实验中，利用gemu 模拟器启动并运行一个虚拟机，以下是详细的实验操作步骤：

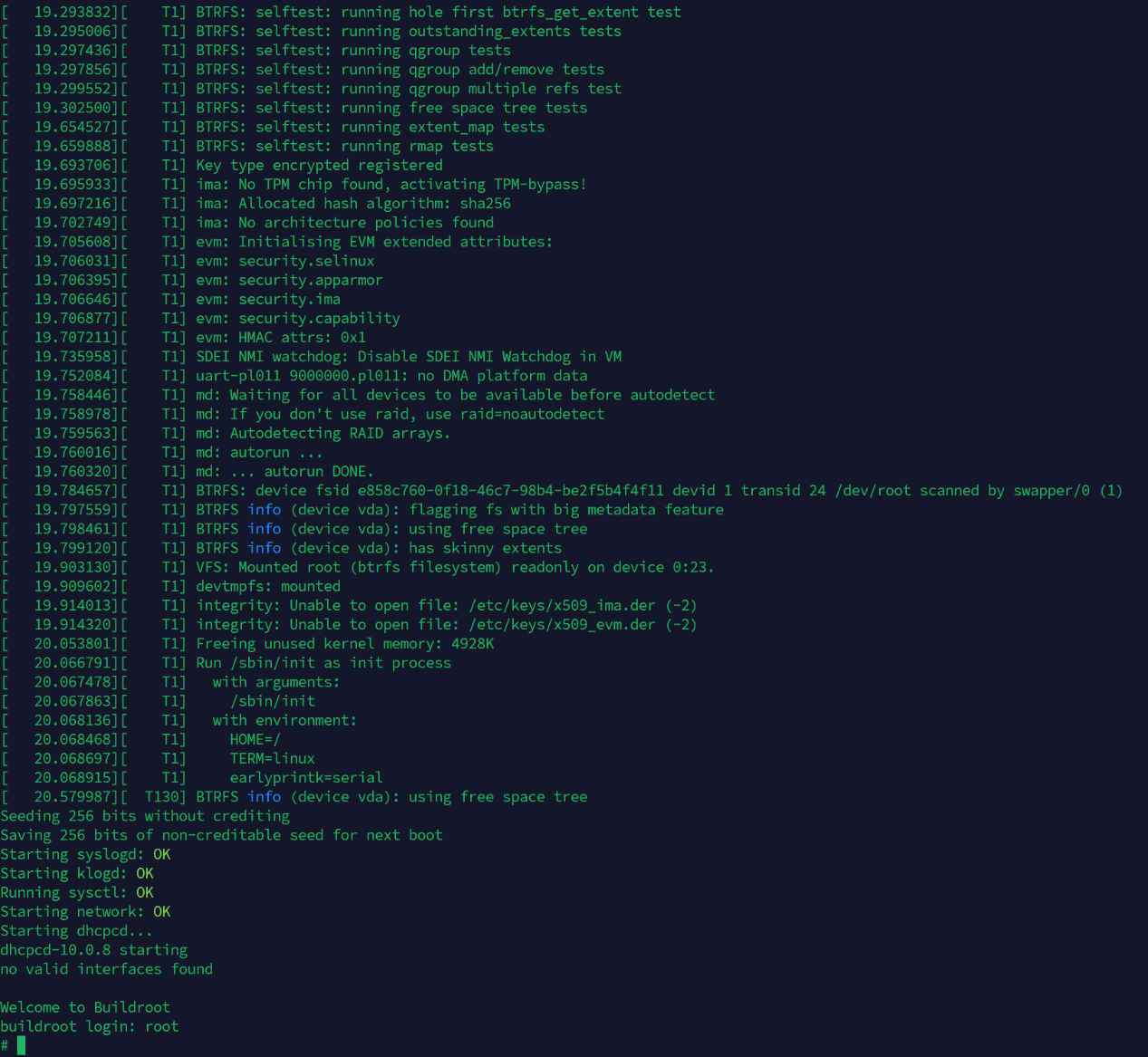
1. 修改Linux内核默认编译配置，移除对ext4文件系统的默认支持，并新增对btrfs 文件系统的支持，随后进行内核编译。
2. 利用 buildroot工具构建根文件系统（rootfs），配置Target options为最终运行该系统的平台类型。配置 Filesystem images为 btrfs 格式，此操作将生成btrfs 类型的虚拟磁盘，内含rootfs。配置Target packages，向rootfs 中添加 vim 和 bash。配置System configuration，将默认shell设置为bash。
3. 通过qemu，添加共享文件夹和edu 设备，结合已编译的内核及虚拟磁盘，尝试启动并运行基于btrfs 虚拟磁盘的虚拟机。
4. 将在qemu虚拟机外编译好的edu驱动程序复制到虚拟机内，并进行安装和运行测试。
5. 使用默认配置的内核，进入虚拟机后查询内核参数kernel.shmmax的值，并尝试修改该参数。修改完毕后，再次查询kernel.shmmax的值，以验证修改是否成功。

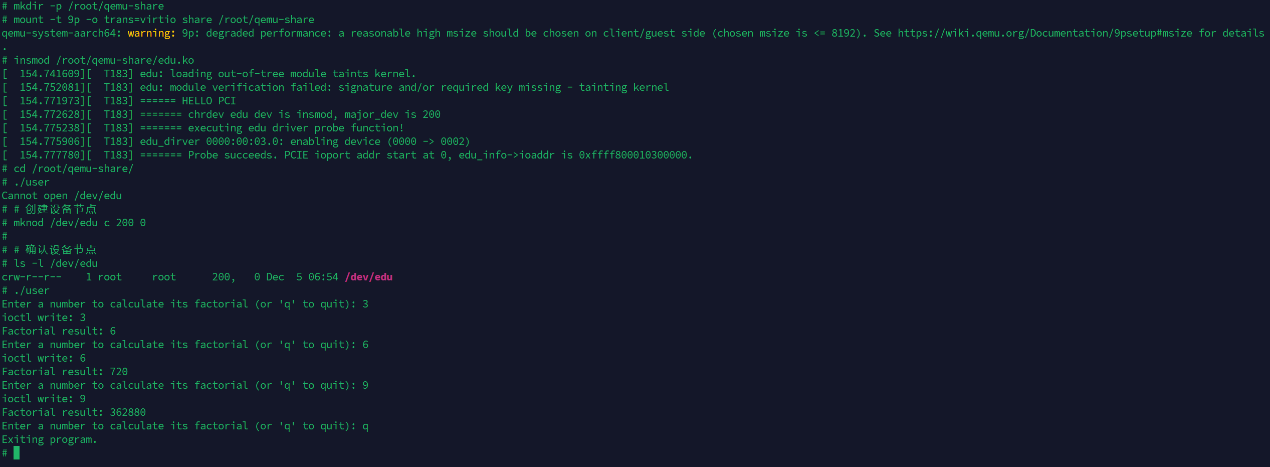
实验步骤

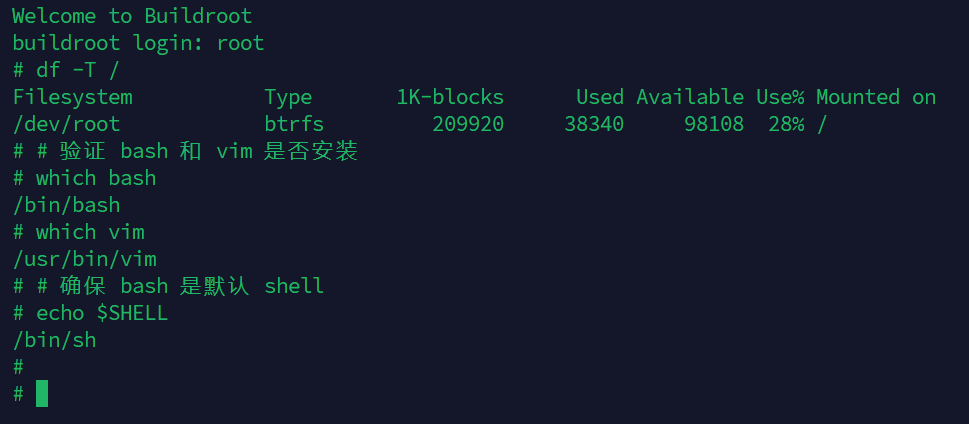
1. 使用主机远程管理连接工具Terminus连接Pod，再ssh连接进入虚拟机，进入/lab/4路径，进入/kernel文件夹，输入make menuconfig打开内核配置界面，取消ext4文件系统支持，勾选btrfs文件系统，保存配置并退出配置界面，输入make -j $(nproc) Image.gz 编译内核。进入buildroot-2024.02.6目录，打开Buildroot配置界面，设置架构类型、文件系统格式、添加工具，然后输入make -j $(nproc)构建根文件系统
2. 创建文件夹/root/share，将实验三中创建的模块文件和测试文件拷贝到此目录下
3. 运行start.sh文件来启动虚拟机，登录时输入root进入系统
4. 在虚拟机中创建文件夹mkdir -p /root/qemu-share，输入mount -t 9p -o trans=virtio share /root/qemu-share挂载共享文件夹
5. 验证安装的工具是否正常
6. 进入共享文件夹装载edu模块并测试
7. 查看内核参数，进行修改并检测
8. 卸载模块、删除挂载点、退出

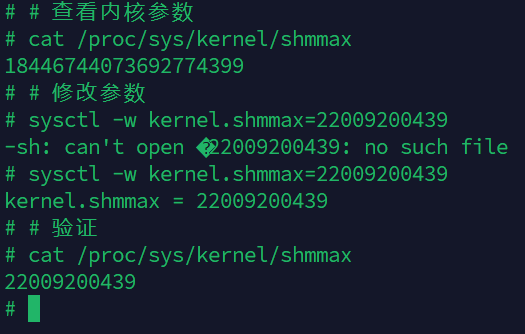
实验结果











关键问题

iv. kernel.shmmax 是 Linux 内核中的一个参数，用于设置共享内存段的最大大小（以字节为单位）调大参数值之后，能够提高Web服务器性能、提高通过共享内存进行通信的效率、提高图形密集型应用场景下数据处理效率

1. 可行，这种做法其实就是交叉编译。现有的交叉编译工具链和文件系统构建工具已经能够允许开发者再一种架构上编译另一种架构的可执行文件

5 个人体会

遇到的主要问题

讲真，这次实验真让我感觉到自己的知识还是匮乏的，不少时候甚至发现某个宏或者函数根本想不到是这样使用的；另外，在调试Bug的时候也遇到不少问题，以前很少使用gcc和g++进行调试，实验过程中的调试非常不习惯

解决办法

1. 查询网络上现有的资料、C++官方标准文档、询问GPT给出意见来进行学习和尝试
2. 在BiliBili等学习平台对命令行调试进行学习
3. 与对底层系统/C++功底好的朋友进行讨论交流