实验1——实验方法

姓名：王越洋 学号：22009200894

**1.实验准备**

**1.1 实验要求**

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1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。

在卸载模块时停止内核线程并释放资源。

**1.2 准备工作**

（1）操作系统：需要openKylin或内核版本为的Linux发行版5.10.0。

（2）开发工具: 安装必要的工具

1.[gcc(GNU 编译器集合)](https://blog.csdn.net/qq_39872252/article/details/133309120)，用于编译内核模块

2.Make使用提供的管理构建Makefile，用于自动化构建模块。

3.安装VS Code来编辑代码。

4.内核头文件：编写内核模块需要内核头文件

（3）内核设置：确保内核模块支持加载功能。

**2.代码结构设计**

**2.1理解关键的内核结构**

（1）内核链表：复习Linux内核提供的链表结构，如list\_head及其相关操作（list\_add, list\_del等）。

（2）内存管理：复习kmalloc()和kfree()，这些函数用于内核内存的动态分配与释放，确保理解如何在内核空间中安全地进行内存管理。

（3）线程与同步：内核线程的创建和停止是通过kthread\_create()和kthread\_stop()实现，而同步机制需要使用spin\_lock。

**2.2 分析代码**

分析现有代码结构，并添加符合实验要求的功能。

**2.3 线程1的实现**

使用for\_each\_process()宏遍历系统中的所有进程，捕获每个进程的pid和进程名，把这些信息添加到自定义的内核链表中。

**技术细节：**通过for\_each\_process()宏遍历系统进程，使用kmalloc()为每个进程节点动态分配内存，存储到自定义链。

**同步处理：**在线程1添加链表节点时，必须使用自旋锁spin\_lock来保护链表操作，[防止与线程2发生竞争](https://blog.csdn.net/EterNity_TiMe_/article/details/142925483)。

**2.4 线程2的实现**

线程2从链表中不断提取节点信息，并使用printk()函数将其打印到内核日志中。

**技术细节：**使用list\_for\_each\_entry\_safe()遍历链表，并使用printk()打印信息。遍历时使用自旋锁保护链表操作，避免与线程1冲突。

**2.5 同步与资源管理**

同步的原理：自旋锁是一种低开销的锁，当一个线程持有锁时，其他线程在尝试获取锁时会自旋等待，直到锁被释放。这样可以确保链表操作的安全性。

技术：为了确保线程1和线程2不会同时操作链表，必须使用自旋锁来同步这两个线程。通过spin\_lock()和spin\_unlock()保证互斥操作。

**2.6 模块的初始化与卸载**

（1）模块初始化

在初始化函数\_\_init my\_module\_init()中，创建并启动两个内核线程，初始化自定义链表和自旋锁。创建线程时，使用kthread\_create()并检查线程创建的返回值是否成功。

（2）模块卸载

在卸载函数\_\_exit my\_module\_exit()中，安全地停止两个内核线程，确保它们执行完毕。遍历链表，释放所有节点的内存，避免内存泄漏。最后，使用rmmod命令卸载模块。

**4. 编译与测试**

**4.1 编写Makefile**

确保Makefile中的配置正确。运行[Make](https://blog.csdn.net/weixin_45031801/article/details/138731712)命令进行编译。如果编译出现错误，可以根据错误提示检查代码的语法或者函数调用。

**4.2 模块加载**

使用以下命令加载模块：

sudo insmod kernel\_module.ko

使用dmesg命令查看内核日志，确认模块是否正确加载，两个线程是否正常运行，进程信息是否被添加到链表中。

**4.3 模块卸载**

使用以下命令卸载模块：

sudo rmmod kernel\_module

卸载后再次查看dmesg，确认内核线程已安全停止，链表资源已释放，内存没有泄漏。

**5. 调试与验证**

**5.1 调试**

（1）查看日志

使用dmesg查看打印信息，确认进程的pid和进程名是否被正确输出。如果未输出，可以检查printk()调用是否正确。

（2）资源管理检查

确保模块卸载后没有残留的内存泄漏或未释放的资源。可以通过多次加载和卸载模块测试内存释放情况。

**5.2 验证**

（1）重复实验

多次加载、卸载模块，确保两个线程正常工作，所有进程信息能够被正确地存储、打印，并且模块能够安全退出。

（2）竞争条件测试

通过增加两个线程操作链表的频率，确保自旋锁能够正常防止竞争条件。

**6.参考链接**

[【Linux】自动化编译工具——make/makefile(超细图例详解！！)](https://blog.csdn.net/weixin_45031801/article/details/138731712)

[【Linux线程】Linux线程编程基础：概念、创建与管理](https://blog.csdn.net/EterNity_TiMe_/article/details/142925483)

[Linux线程——线程创建和基本使用（多线程并发）](https://blog.csdn.net/yi_chengyu/article/details/120412980)

[Gcc使用](https://blog.csdn.net/qq_39872252/article/details/133309120)