操作系统实验设计：edu 设备驱动

1. 实验题目

1.1 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。

1.2 实验内容

本次实验旨在让学生深入理解并实践 edu 设备驱动的开发。实验中，我们将提供 edu 设备驱动的框架代码，学生需在此基础上完成关键代码的实现。具体实验要求如下：

1. 补全框架中的 TODO 位置的缺失代码，包含 TODO 的函数如下所示：
    - 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
    - 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
    - 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
    - edu_dev_open
        - 完成 filp 与用户进程上下文信息的绑定操作
    - edu_dev_release
        - 释放 edu_dev_open 中分配的内存
    - 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
2. 实现驱动程序的 ioctl 调用处理功能。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。

3. 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。
4. 编写 C 语言应用程序，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1，再次调用 ioctl 接口，以获取设备计算完成的结果。

2 实验代码框架解析

2.1 驱动加载流程

在成功利用 insmod 指令装载驱动后，系统将自动触发 edu_driver_init 函数的执行，以完成驱动程序的初始化工作。在 edu_driver_init 函数的执行流程中，首先通过调用 register_chrdev 函数来注册一个字符设备，该过程需要指定主设备号、设备名称，以及与设备操作相关的结构体 edu_dev_fops。

- 当通过 register_chrdev 函数在内核中注册字符设备之后，便可在用户空间中使用 mknod 命令创建相应的设备节点。这样一来，用户空间的程序便能够通过这一设备文件与底层设备驱动进行交互。
- register_chrdev 函数赋予了驱动程序开发者定义文件操作集合（struct file_operations）的能力，该集合包含了诸如 open、read、write、close、ioctl 等操作，这些操作与用户空间程序对设备文件发起的系统调用相对应。

随后，edu_driver_init 函数将执行 pci_register_driver 调用，将 PCI 设备驱动程序注册至内核的 PCI 子系统。在调用 pci_register_driver 函数时，必须提供一个 struct pci_driver 类型的参数。该结构体封装了 PCI 设备驱动的相关信息，其中包含的设备发现函数，其会在 PCI 设备驱动注册成功后由 Linux 内核调用自动执行，主要职责是对相应的 PCI 设备进行初始化。在本框架中，edu_driver_probe 函数即为设备发现函数。该函数内部首先通过调用 pci_enable_device 函数来启用 PCI 设备，并为其分配必要的资源。随后，需利用 edu_dev_info 结构体来存储 edu 设备的相关信息。请遵循 TODO 提示，为 edu_dev_info 的实例分配适当的内存空间。紧接着，通过 pci_request_regions 函数请求 PCI 设备的 I/O 和内存资源区域，以防止这些资源被其他设备或驱动程序占用。接下来，要将 edu 设备的基地址寄存器（Base Address Register, BAR）映射到驱动程序的地址空间，以便驱动程序能够通过读写这些映射后的虚拟地址来访问 PCI 设备的资源，请根据 TODO 提示，完成将 BAR 的总线地址映射到系统内存的虚拟地址的操作。在 edu_driver_probe 函数的最后部分，使用 pci_set_drvdata 函数来设置驱动的私有数据，将 edu_dev_info 实例与设备结构体 dev 关联起来。如此一来，便可以通过 dev 指针来访问 edu_dev_info 实例。至此，驱动的初始化完成。

2.2 驱动注销流程

当通过 rmmod 命令卸载驱动程序时，系统将自动触发调用 edu_driver_exit 函数。在此函数中，首先执行的操作是调用 unregister_chrdev，以此注销字符设备。随后，通过调用 pci_unregister_driver，将 PCI 设备驱动程序从内核的 PCI 子系统中注销。在 PCI 设备注销过程中，edu_driver_remove 函数会被自动调用，负责进行设备的清理工作。在此函数中，首先需要从设备结构体 dev 中提取 edu_dev_info 实例，请遵循 TODO 指示完成这一步骤。接下来，根据 TODO 提示，补全 iounmap 函数的调用，以取消 BAR 区域的总线地址映射。然后，调用 pci_release_regions 以释放 PCI 设备的 I/O 和内存资源区域。紧接着，请根据 TODO 提示，释放 edu_dev_info 实例所占用的内存资源。最终，通过调用 pci_disable_device 函数，关闭 PCI 设备，完成整个卸载过程。

2.3 用户使用流程

在用户进程操作设备之前，必须首先通过 open 系统调用对目标设备进行打开操作。得益于驱动加载流程中 register_chrdev 字符设备注册，open 操作将触发调用 edu_dev_open 函数。在本案例中，edu_dev_open 的主要职责是保存用户空间调用的上下文信息，这样做是为了能够区分不同进程的调用，并确保计算结果能准确返回给相应的调用进程。请遵循 TODO 提示，完成 filp 与上下文信息的绑定操作。

当用户进程成功执行 open 操作并收到返回值后，将获得设备的文件描述符。此时，只需将文件描述符作为参数传递给 ioctl 系统调用，便可实现对设备的操作。ioctl 操作将调用 edu_dev_unlocked_ioctl 函数，在本案例中，用户空间程序需要通过 ioctl 传入要计算阶乘的数值，并且通过 ioctl 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，在线程 kthread_handler 内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。请根据 TODO 提示，完善 edu_dev_unlocked_ioctl 与 kthread_handler 函数的实现。

在完成设备使用后，用户进程需通过 close 系统调用关闭设备的文件描述符。close 操作将引发 edu_dev_release 函数的调用。在本案例中，edu_dev_release 负责释放 edu_dev_open 函数中分配的内存资源。请依据 TODO 提示，完成相应的清理工作。

3. 实验环境及平台

- 操作系统：openKylin
- 内核版本：Linux 5.10.0
- 处理器数量：2
- 内存：4GB

4. 实验前置要求

本实验需要了解字符设备驱动、内核线程、pci 设备相关知识。

5. API 介绍

5.1 内核线程

内核线程是直接由内核本身启动的进程。内核线程实际上是将内核函数委托给独立的进程，它与内核中的其他进程”并行”执行。内核线程经常被称之为内核守护进程 。

5.1.1 创建内核线程

宏 kthread_create(threadfn, data, namefmt, arg...)

说明：

- 创建一个内核线程，并返回其句柄。（创建好的线程不会直接运行，需要调用 wake_up_process）

参数：

- threadfn：线程执行体。需要是一个返回值为 int，参数为 void*的函数。
- data：传入到线程执行体中的数据。
- namefmt：线程名。

返回值：

- 内核线程句柄。

5.1.2 唤醒休眠线程

int wake_up_process(struct task_struct *tsk)

说明：

- 用于唤醒处于睡眠状态的进程，使进程由睡眠状态变为 RUNNING 状态，从而能够被 CPU 重新调度执行。

参数：

- tsk：内核线程句柄

返回值：

- 1：唤醒成功
- 0：唤醒失败

5.1.3 内核线程的退出

int kthread_stop(struct task_struct *thread);

说明：

- 向目标内核线程发送退出信号，等待其退出。

参数：

- thread：内核线程句柄

返回值：

- 返回线程执行体的结果，如果从未调用 wake_up_process()，则返回 -EINTR。

5.2 PCI 设备

5.2.1 初始化 PCI 设备

int pci_enable_device(struct pci_dev * dev);

说明：

- 初始化 PCI 设备，请求底层代码启用 IO 与内存。

参数：

- dev：需要被初始化的设备

返回值：

- 0：执行成功
- 其他：执行失败

5.2.2 保留 PCI 区域

int pci_request_regions(struct pci_dev *dev, const char *res_name);

说明：

- 保留 PCI 资源，防止其他 PCI 设备使用这些资源。

参数：

- dev：指向 PCI 设备的指针
- res_name：资源的名称

返回值：

- 0：执行成功
- 其他：执行失败

5.2.3 映射 BAR 总线地址

void *pci_ioremap_bar(struct pci_dev *pdev, int bar);

说明：

- 将 PCI 设备 BAR 总线地址映射到内核的地址空间，允许内核直接访问 PCI 设备的内存或 I/O 空间。

参数：

- pdev：指向 PCI 设备指针
- bar：要映射的基地址寄存器（BAR）的索引

返回值：

- 指向映射后的内存区域的指针，失败返回 NULL

5.2.4 设置驱动私有数据

void *pci_set_drvdata(struct pci_dev *pdev, void *data);

说明：

- 将私有数据与一个 PCI 设备结构体关联起来。

参数：

- pdev：指向 PCI 设备的指针。
- data：指向要设置的私有数据的指针。

返回值：

- 返回指向之前与该设备关联的私有数据。如果之前没有设置过私有数据，则返回 NULL。

5.2.5 取消地址映射

void iounmap(void __iomem *addr);

说明：

- 解除之前通过 ioremap 或类似函数创建的 I/O 地址空间到内核虚拟地址空间的映射。

参数：

- addr:指向之前通过 ioremap 映射的虚拟地址空间的指针。

返回值：无

5.2.6 释放 PCI 区域

void pci_release_regions (struct pci_dev * pdev);

说明：

- 释放保留的 PCI I/O 和内存资源

参数：

- pdev：指向 PCI 设备的指针

返回值：无

5.2.7 注销 PCI 设备

void pci_disable_device(struct pci_dev * dev);

说明：

- 向系统发出 PCI 设备不再被系统使用的信号。

参数：

- dev：指向 PCI 设备的指针。

返回值：无

5.2.8 读取 PCI 设备数据

unsigned int readl (unsigned int addr);

说明：

- 从内存映射的 I/O 空间读取数据

参数：

- addr：IO 地址

返回值：

- 读取到的数据

5.2.9 写入 PCI 设备数据

void writel(unsigned int data, unsigned int addr);

说明：

- 向内存映射的 I/O 空间上写数据。

参数：

- data：要写入的一个字节的数据。
- addr：I/O 地址。

返回值：无。

5.2.10 注册 PCI 驱动

int pci_register_driver(struct pci_driver * drv);

说明：

- 将驱动程序结构添加到已注册的驱动程序列表中

参数：

- drv：待注册的驱动程序结构

返回值：

- 0：成功
- 其他：失败

5.2.11 注销 PCI 驱动

int pci_unregister_driver(struct pci_driver * drv);

说明：

- 将驱动程序结构从已注册的驱动程序列表中移除

参数：

- drv：待移除的驱动程序结构

返回值：

- 0：成功
- 其他：失败

5.3 内核内存管理

5.3.1 kmalloc

void *kmalloc(size_t size, gfp_t flags);

说明：

- kmalloc 用于在内核空间分配内存。

参数：

- size：请求分配的内存大小。
- flags：分配标志，用于指定内存分配的行为和属性
  - GFP_KERNEL：普通的内核内存分配，可能会休眠。
  - GFP_ATOMIC：原子内存分配，不会休眠，常用于中断上下文。

返回值：

- 被分配的内存地址。

5.3.2 kfree

void kfree(const void *objp);

说明：

- kfree 用于释放由 kmalloc 分配的内存。

参数：

- objp：指向要释放的内存块的指针。

返回值：无

5.4 字符设备

5.4.1 注册字符设备

int register_chrdev(unsigned int major, const char *name, const struct file_operations *fops);

说明：

- 在内核中注册一个新的字符设备

参数：

- major：设备主设备号。如果设置为0，内核会动态分配一个主设备号。

- name：设备名称。
- fops：指向 file_operations 结构体的指针，该结构体包含了设备操作的函数指针，如打开、读取、写入等。

返回值：

- 0：执行成功
- 其他：执行失败

5.4.2 注销字符设备

void unregister_chrdev(unsigned int major, const char *name);

说明：

- 注销之前通过 register_chrdev 注册的字符设备

参数：

- major：注销设备的主设备号。
- name：注销设备的名称。

返回值：无。

5.5 原子操作

5.5.1 设置原子变量

void atomic_set(atomic_t *v, int i)

说明：

- 原子的设置 v 的值

参数：

- v：指向原子变量的指针
- i：设定值

返回值：无

5.5.2 读取原子变量

int atomic_read(const atomic_t * v);

说明：

- 原子的读取 v 的值

参数：

- v：指向原子变量的指针

返回值：读取的原子变量值