操作系统实验手册：deferred work

1. 实验内容

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的 workqueue 技术，理解其与 kernel thread 的区别。

1.2 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。

具体实验步骤如下：

1. 分别采用 work queue 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1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10个不同的 kernel thread 分别执行。
2. 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。

2. 实验结果

使用 dmesg 命令可以观察到 work queue 的函数按顺序执行，而内核线程的函数执行不是顺序的。模块加载五秒后，显示 delayed work 的输出。

```
[  673.258500] deferred work module init
[  673.258854] work queue : 315
[  673.258855] work queue : 316
[  673.258856] work queue : 317
[  673.258856] work queue : 318
[  673.258856] work queue : 319
[  673.258857] work queue : 320
[  673.258857] work queue : 321
[  673.258857] work queue : 322
[  673.258858] work queue : 323
[  673.258858] work queue : 324
[  673.260184] kthread : 316
[  673.260530] kthread : 317
[  673.260575] kthread : 315
[  673.261025] kthread : 318
[  673.261421] kthread : 319
[  673.261996] kthread : 320
[  673.262586] kthread : 321
[  673.263107] kthread : 322
[  673.263505] kthread : 323
[  673.266438] kthread : 324
[  678.453724] delayed work!
```

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 deferred work、work queue、内核线程相关知识。

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 work queue

5.2.1 初始化 work_struct

宏 INIT_WORK(struct work_struct *work, void (*function)(struct work_struct *work))

说明：

- 初始化 work 句柄，将执行例程与 work 绑定。

参数：

- work：work 句柄。
- function：work 的执行例程。

返回值：无

5.2.2 调度 work

bool schedule_work(struct work_struct * work)

说明：

- 将 work 提交给全局的 work queue，负责执行完 work 的程序

参数：

- work：work 句柄。

返回值：

- bool：是否提交成功

5.2.3 初始化 delayed_work

宏 INIT_DELAYED_WORK(struct delayed_work *dwork, void (*function)(struct work_struct *work))

说明：

- 初始化 dwork 句柄，将执行例程与 dwork 绑定。

参数：

- dwork：delayed work 句柄。
- function：delayed work 执行例程。

返回值：无

5.2.4 调度 delayed work

bool schedule_delayed_work(struct delayed_work *dwork, unsigned long delay)

说明：

- 将 delayed work 提交给全局的 work queue。它和 schedule_work 的区别在于有延时。

参数：

- dwork：delayed work 句柄。
- delay：延迟时间，单位为 jiffies。

返回值：

- bool：是否提交成功

5.3 container_of

宏 container_of(ptr, type, member)

说明：

- 通过结构体成员变量地址获取目标结构体的地址

参数：

- ptr：结构体成员地址
- type：目标结构体类型
- member：结构体成员名称

返回值：

- 目标结构体地址

5.4 内核休眠

unsigned long msleep_interruptible(unsigned int msecs)

说明：

- 可以被信号唤醒的休眠。

参数：

- msecs：休眠时长，单位 ms

返回值：

- 休眠的时长。