实验项目：同步互斥

姓名：张伟焜　学号：17343155　邮箱：zhangwk8@mail2.sysu.edu.cn
院系：数据科学与计算机学院 专业：17 级软件工程 指导教师：张永东

【实验题目】

同步互斥

【实验目的】

理解操作系统的同步互斥的设计实现；
理解底层支撑技术：禁用中断、定时器、等待队列；
在 ucore 中理解信号量（semaphore）机制的具体实现；
了解经典进程同步问题，并能使用同步机制解决进程同步问题。

【实验要求】

根据指导，完成练习 0~1。

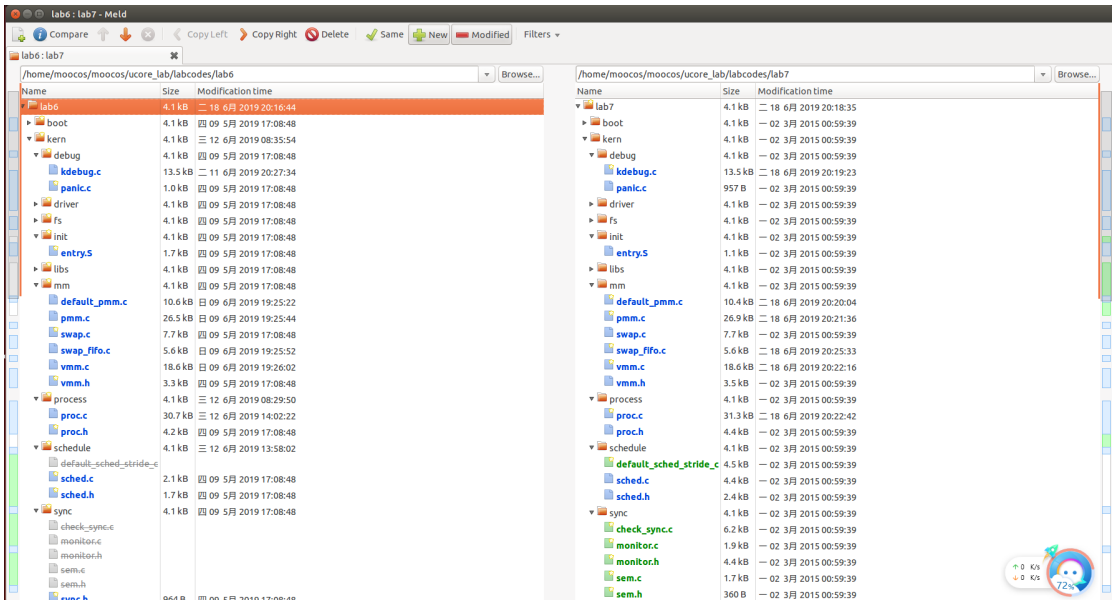
【实验方案】

实验环境：老师提供的虚拟机（Virtual box），无特殊硬件要求
实验思路：根据实验指导，先了解理论知识，再进行实验

【实验过程】

练习 0：填写已有实验。

使用 meld 软件将 ucore 启动实验的代码导入。



注意要点击标星文件进行对比，将上次实验完成的函数复制过来，不要将整个文件进行覆盖。之前实验修改的内容主要在 kdebug.c trap.c pmm.c default_pmm.c vmm.c swap_fifo.c proc.c 等。

修改 trap.c 中的 trap_dispatch 函数。将 sched_class_proc_tick 函数换为 run_timer_list 函数，用于支持定时器机制。

```
ticks++;
assert(current != NULL);
sched_class_proc_tick(current);
run_timer_list();
break;
```

练习 1：理解内核级信号量的实现和基于内核级信号量的哲学家就餐问题。

完成练习 0 后，建议大家比较一下（可用 meld 等文件 diff 比较软件）个人完成的 lab6 和练习 0 完成后的刚修改的 lab7 之间的区别，分析了解 lab7 采用信号量的执行过程。执行 make grade，大部分测试用例应该通过。

分析 lab7 采用信号量的执行过程。

首先触发 sync/check_sync.c 中的 check_sync 函数。结合注释，check_sync 函数中第一个循环使用信号量实现哲学家就餐问题，第二个循环利用条件变量，即管程的方式实现哲学家就餐问题。

分析采用信号量实现的部分如下。sem_init 函数首先初始化了一个全局的互斥信号量，然后在循环中创建了 N 个信号量代表 N 个哲学家的行为。

```
//check semaphore
    sem_init(&mutex, 1);
    for (i = 0; i < N; i++) {
        sem_init(&s[i], 0);
        int pid = kernel_thread(philosopher_using_semaphore, (void *)i, 0);
        if (pid <= 0) {
            panic("create No.%d philosopher_using_semaphore failed.\n");
        }
        philosopher_proc_sema[i] = find_proc(pid);
        set_proc_name(philosopher_proc_sema[i], "philosopher_sema_proc");
    }
```

此后调用 philosopher_using_semaphore 函数。philosopher_using_semaphore 函数为每个哲学家创建一个内核线程。

```
int philosopher_using_semaphore(void * arg) /* i：哲学家号码，从0到N-1 */
{
    int i, iter = 0;
    i = (int)arg;
    cprintf("I am No.%d philosopher_sema\n", i);
    while (iter++ < TIMES)
    { /* 无限循环 */
        cprintf("Iter %d, No.%d philosopher_sema is thinking\n", iter, i); /* 哲学家正在思考 */
        do_sleep(SLEEP_TIME);
        phi_take_forks_sema(i);
        /* 需要两只叉子，或者阻塞 */
        cprintf("Iter %d, No.%d philosopher_sema is eating\n", iter, i); /* 进餐 */
        do_sleep(SLEEP_TIME);
```

```
        phi_put_forks_sema(i);
        /* 把两把叉子同时放回桌子 */
    }
    cprintf("No.%d philosopher_sema quit\n", i);
    return 0;
}
```

拿叉子与放叉子的函数如下:

```
void phi_take_forks_sema(int i) /* i：哲学家号码从0到N-1 */
{
    down(&mutex); /* 进入临界区 */
    state_sema[i] = HUNGRY; /* 记录下哲学家i饥饿的事实 */
    phi_test_sema(i); /* 试图得到两只叉子 */
    up(&mutex); /* 离开临界区 */
    down(&s[i]); /* 如果得不到叉子就阻塞 */
}

void phi_put_forks_sema(int i) /* i：哲学家号码从0到N-1 */
{
    down(&mutex); /* 进入临界区 */
    state_sema[i] = THINKING; /* 哲学家进餐结束 */
    phi_test_sema(LEFT); /* 看一下左邻居现在是否能进餐 */
    phi_test_sema(RIGHT); /* 看一下右邻居现在是否能进餐 */
    up(&mutex); /* 离开临界区 */
}
```

执行 make grade。

```
  -check output:                          OK
sleep:                  (11.8s)
  -check result:                          OK
  -check output:                          OK
sleepkill:              (3.2s)
  -check result:                          OK
  -check output:                          OK
matrix:                 (11.9s)
  -check result:                          OK
  -check output:                          OK
Total Score: 183/190
make: *** [grade] Error 1
zhangweikun$>
```

请在实验报告中给出内核级信号量的设计描述，并说其大致执行流流程。

在 sem.h 中定义了信号量的结构体。结构体中包括信号量的值和一个等待队列。

```
 8 typedef struct {
 9     int value;
10     wait_queue_t wait_queue;
11 } semaphore_t;
```

sem_init 函数对信号量进行初始化。该函数只需要将信号量设置为指定初始值，并且将等待队列初始化即可。

```
void
sem_init(semaphore_t *sem, int value) {
    sem->value = value;
    wait_queue_init(&(sem->wait_queue));
}
```

等待队列的代码在 kern/sync/wait.c 和 wait.h 中。等待队列的操作分为两层，底层函数是对 wait_queue 的初始化、插入、删除、查找操作。高层函数基于底层函数实现了让进程进入等待队列以及唤醒进程的功能。

P 和 V 函数。P 函数由“down(semaphore_t *sem)”实现，V 函数由“up(semaphore_t *sem)”实现，具体实现是“__down(semaphore_t *sem, uint32_t wait_state)”函数和“__up(semaphore_t *sem, uint32_t wait_state)”函数。

__up 函数首先关中断，若信号量对应的 wait queue 中没有在等待的进程（等待队列为空）就对信号量的值加一，然后开中断返回。

如果等待队列不为空，就判断等待的进程的等待原因是否是信号量设置的，然后调用 wakeup_wait 函数将等待的第一个进程唤醒，之后开中断返回。该函数对应到了原理课中提及到的 V 操作，表示释放了一个该信号量对应的资源，如果有等待在了这个信号量上的进程，则将其唤醒执行；结合函数的具体实现可以看到其采用了禁用中断的方式来保证操作的原子性。

```
static __noinline void __up(semaphore_t *sem, uint32_t wait_state) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        wait_t *wait;
        if ((wait = wait_queue_first(&(sem->wait_queue))) == NULL) {
            sem->value++;
        }
        else {
            assert(wait->proc->wait_state == wait_state);
            wakeup_wait(&(sem->wait_queue), wait, wait_state, 1);
        }
    }
    local_intr_restore(intr_flag);
}
```

__down 函数被触发后关中断，判断是否存在多余的可分配的资源，是的话取出资源（整型变量减 1），然后返回并开中断。这里关中断保证了在修改信号量的时候不会出现中断现象，即原子操作。如果不能获得信号量的话（没有可用的资源）就把当前进程加入等待队列中，接着开中断后，调用 schedule 函数来让出 CPU，在资源得到满足，重新被唤醒之后，将自身从等待队列上删除掉。该函数对应原理课中提及的 P 操作，表示请求一个该信号量对应的资源。

```
static __noinline uint32_t __down(semaphore_t *sem, uint32_t wait_state) {
    bool intr_flag;
    local_intr_save(intr_flag);
```

```
    if (sem->value > 0) {
        sem->value--;
        local_intr_restore(intr_flag);
        return 0;
    }
    wait_t __wait, *wait = &__wait;
    wait_current_set(&(sem->wait_queue), wait, wait_state);
    local_intr_restore(intr_flag);

    schedule();

    local_intr_save(intr_flag);
    wait_current_del(&(sem->wait_queue), wait);
    local_intr_restore(intr_flag);

    if (wait->wakeup_flags != wait_state) {
        return wait->wakeup_flags;
    }
    return 0;
}
```

请在实验报告中给出给用户态进程/线程提供信号量机制的设计方案，并比较说明给内核级提供信号量机制的异同。

用户态信号量可以用内核态信号量实现。

操作系统内核提供关于信号量的系统调用，当用户进程需要对信号量进行操作时，通过系统调用进入内核态，在内核态中进行信号量操作。由于一个用户进程可能涉及到多个内核临界区，所以用户进程信号量结构体要包括一个信号量数组，用于存放不同临界区的信号量。

设计方案：

1.定义信号量结构体：一个由内核信号量结构体组成的 vector。

2.定义生成信号量的函数 GenerateSemaphore()，用于创建信号量数组并初始化。

3.定义信号量增加和减去操作：直接调用内核__up 和__down 函数。

用户态进程/线程提供信号量机制与内核级提供信号量机制相同点：

两者的基本实现逻辑相同

用户态进程/线程提供信号量机制与内核级提供信号量机制不同点：

在用户态下实现信号量机制涉及到操作系统的系统调用的支持；而在内核态下实现不需要。

【实验总结】

完成实验后，请分析 ucore_lab 中提供的参考答案，并请在实验报告中说明你的实现与参考答案的区别。

本次实验不涉及编程所以没有参考答案。

列出你认为本实验中重要的知识点，以及与对应的 OS 原理中的知识点，并简要说明你对二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知识点）

1.底层为操作系统实现互斥访问的机制（禁用中断、定时器、等待队列、Test and Set 指令等）；

2.信号量的设计；

3.哲学家就餐问题，解决同步互斥问题的方法；

4.进程的调度；

对应了 OS 原理中的：

1.系统中断的处理

2.具体实现信号量机制、条件变量和管程机制的方法；

3.进程同步互斥问题及具体解决同步互斥问题的实现；

理论知识是基础；

实验知识是理论知识的实际应用与实践。

列出你认为 OS 原理中很重要，但在实验中没有对应上的知识点

1.原子事务中的基于日志的恢复

2.没有分析管程

心得体会

很高兴本次实验没有编程任务。本次实验围绕着同步与互斥展开，并结合哲学家就餐问题进行深入的分析。

通过练习 1 的相关分析，我对内核级信号量的设计描述及其大致执行流程有了深入地了解，并且掌握了内核级提供信号量机制的异同。

本次实验的内容不多，但通过练习，我加深了对理论课涉及的相关知识的认识，有助于知识的巩固。

【参考文献】

《操作系统实验指导(清华大学)陈渝、向勇编著》