

Lab7

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（不需要编码）

1.1 Lab7 内容

1. 代码结构：

- Lab7 在 `kern` 目录下新增了 `sync` 目录，其中包含了 `sem.c`, `sem.h` (信号量实现), `monitor.c`, `monitor.h` (管程实现), `check_sync.c`。

2. 进程控制块 (`proc_struct`)：

- Lab7 为了支持信号量机制，在 `proc_struct` 中使用了 `wait_state` 字段，并定义了新的等待状态 `WT_KSEM` (等待内核信号量)。虽然 `wait_state` 可能在之前就存在，但其用途在 Lab7 中被扩展到了同步原语的等待。

3. 系统功能：

- Lab7 的核心功能是实现进程/线程间的同步与互斥，引入了信号量和管程机制，解决了并发执行中的资源竞争问题。

4. 执行过程分析：

- 在 Lab7 中，当一个内核线程执行信号量的 `down` 操作且资源不足时，它不会像忙等待那样消耗 CPU，而是将自己的状态设置为 `WT_KSEM`，加入到信号量的等待队列中，并调用 `schedule()` 主动让出 CPU。这需要调度器 (Lab6 的成果) 的支持来选择下一个可运行的进程。
- 当资源被释放 (`up` 操作) 时，处于等待队列的进程会被唤醒 (状态变为 `PROC_RUNNABLE`)，并由调度器在未来某个时刻调度执行。

1.2 内核级信号量的设计描述

内核级信号量的数据结构定义在 `kern/sync/sem.h` 中：

```
1 | typedef struct {
2 |     int value;
3 |     wait_queue_t wait_queue;
4 | } semaphore_t;
```

- `value`：信号量的计数值。如果大于 0，表示当前可用的资源数量；如果小于等于 0，表示当前没有可用资源，且 `abs(value)` 可能表示等待该资源的进程数 (取决于具体实现，`ucore` 中 `value` 减到负数表示有等待者，但在 `down` 实现中一旦阻塞就返回了，没有继续减 `value`，见下文分析)。
- `wait_queue`：等待队列，用于悬挂因为无法获得资源而阻塞的进程。

大致执行流程：

1. **初始化** (`sem_init`)：设置 `value` 的初值，并初始化等待队列为空。

2. **P 操作** (`down`)：

- 首先关中断，保证原子性。
- 检查 `sem->value > 0`：

- 若成立，则 `sem->value--`，获得资源，恢复中断并返回。
- 若不成立 (`sem->value <= 0`)，当前进程需要等待。
- 如果需要等待：
 - 将当前进程加入 `sem->wait_queue`。
 - 将当前进程状态设置为 `WT_KSEM`。
 - 调用 `schedule()` 进行调度，切换到其他进程。
- 当进程被唤醒后，从等待队列中移除，恢复中断。

3. V 操作 (up):

- 首先关中断。
- 检查等待队列 `sem->wait_queue` 是否为空：
 - 若为空，表示没有进程在等待，直接 `sem->value++`，释放资源。
 - 若不为空，表示有进程在等待。此时并不增加 `value`（或者说增加后立即被唤醒的进程消耗掉了），而是直接从等待队列中唤醒一个进程 (`wakeup_wait`)。被唤醒的进程将进入 `PROC_RUNNABLE` 状态，等待调度运行。

1.3 哲学家就餐问题与死锁证明

实现逻辑：

- 每个哲学家有一个信号量 `s[i]`（初值为 0），用于在没有叉子时阻塞。
- 有一个互斥信号量 `mutex`（初值为 1），用于保护对状态数组 `state_sema` 的访问。
- `phi_take_forks_sema(i)` 函数流程：
 1. 获取 `mutex`。
 2. 将自己状态设为 `HUNGRY`。
 3. 调用 `phi_test_sema(i)`：检查**左右邻居**是否都在 `EATING`。如果都不是，则自己状态设为 `EATING`，并对 `s[i]` 执行 `up` 操作。
 4. 释放 `mutex`。
 5. 对 `s[i]` 执行 `down` 操作。如果刚才 `test` 成功，`s[i]` 为 1，`down` 不会阻塞；否则 `s[i]` 为 0，哲学家阻塞。

为什么不会出现死锁？

死锁的四个必要条件是：互斥、请求与保持、不剥夺、循环等待。在该实现中，**请求与保持**条件被破坏了。

- 哲学家只有在**同时**拿到两把叉子（即左右邻居都不在进餐）时，才会开始进餐（状态变为 `EATING` 并通过 `s[i]`）。
- 如果无法同时拿到两把叉子，哲学家会进入阻塞状态，但此时他**并没有持有一把叉子而等待另一把**。他是在什么都没有持有的情况下等待。
- 因为对状态的检查和修改是在 `mutex` 保护的临界区内进行的，这是一个原子操作。哲学家要么拿到了两把叉子，要么一把没拿。
- 因此，不存在“哲学家 A 拿了左手叉子等待右手叉子，而哲学家 B 拿了右手叉子等待左手叉子”的情况，从而避免了死锁。

1.4 用户态信号量机制设计方案

给用户态进程/线程提供信号量机制，需要通过系统调用将内核的信号量功能暴露出来。

设计方案：

1. 系统调用接口：

- `int sys_sem_init(int value)`：创建一个新的信号量，初值为 `value`。返回一个信号量 ID (`sem_id`)。
- `int sys_sem_wait(int sem_id)`：对指定 ID 的信号量执行 P 操作。
- `int sys_sem_post(int sem_id)`：对指定 ID 的信号量执行 V 操作。
- `int sys_sem_free(int sem_id)`：释放/销毁指定 ID 的信号量。

2. 内核数据结构：

- 在内核中维护一个全局的信号量数组或链表，或者在每个进程的控制块 (`proc_struct`) 中维护一个该进程打开的信号量表（类似文件描述符表）。
- 如果是全局数组，可以定义：

```
1 struct sem_control_block {  
2     semaphore_t sem;  
3     int owner_pid; // 可选，用于资源回收  
4     bool used;  
5 } sem_array[MAX_SEMS];
```

- 用户态拿到的 `sem_id` 即为该数组的索引。

3. 执行流程：

- 用户调用 `sys_sem_wait(sem_id)`。
- 陷入内核，系统调用处理函数检查 `sem_id` 合法性。
- 内核调用 `down(&sem_array[sem_id].sem)`。
 - 如果需要阻塞，内核将当前用户进程状态设为等待，并调度其他进程。
 - 当被唤醒后，系统调用返回用户态。

内核级与用户态信号量机制的异同：

特性	内核级信号量 (Kernel Semaphore)	用户态信号量 (User Semaphore)
使用对象	内核线程 / 内核代码	用户进程 / 用户线程
访问方式	直接函数调用 (<code>up / down</code>)，直接访问内存地址	系统调用 (System Call)，通过 ID/句柄访问
存储位置	内核栈或内核全局数据区	内核空间 (用户态仅持有句柄)
原子性保证	关中断 (<code>local_intr_save</code>) 或自旋锁	陷入内核后由内核机制 (关中断/锁) 保证
开销	较小，仅涉及函数调用和可能的调度	较大，涉及用户态/内核态切换的上下文开销

特性	内核级信号量 (Kernel Semaphore)	用户态信号量 (User Semaphore)
安全性	开发者完全可控，但也容易导致内核崩溃	需要内核进行严格的参数检查和权限控制，防止用户滥用导致内核问题

练习2 完成内核级条件变量和基于内核级条件变量的哲学家就餐问题（需要编码）

如下是`check_sync.c`文件中我们编写的哲学家就餐问题相关的代码，下面将使用注释的形式解释。

```

1 void phi_take_forks_condvar(int i) {
2     down(&(mtp->mutex));
3 //-----into routine in monitor-----
4     // LAB7 EXERCISE1: YOUR CODE
5     // I am hungry
6     // try to get fork
7     // 设置自己为饥饿状态
8     state_condvar[i] = HUNGRY;
9     // 测试能否就餐
10    phi_test_condvar(i);
11    // 如果不能就餐，等待条件变量
12    while(state_condvar[i] != EATING) {
13        cond_wait(&mtp->cv[i]);
14    }
15 //-----leave routine in monitor-----
16    // 离开管程
17    if(mtp->next_count>0)
18        up(&(mtp->next));
19    else
20        up(&(mtp->mutex));
21 }
22
23 void phi_put_forks_condvar(int i) {
24     down(&(mtp->mutex));
25
26 //-----into routine in monitor-----
27     // LAB7 EXERCISE1: YOUR CODE
28     // I ate over
29     // test left and right neighbors
30     // 设置自己为思考状态
31     state_condvar[i] = THINKING;
32     // 检查左右邻居是否能就餐
33     phi_test_condvar(LEFT);
34     phi_test_condvar(RIGHT);
35 //-----leave routine in monitor-----
36     // 离开管程
37     if(mtp->next_count>0)
38         up(&(mtp->next));
39     else
40         up(&(mtp->mutex));
41 }
```

经过上面的修改后，又对`monitor.c`中的内容进行了修改，下面将使用注释的形式解释。

```
1 // unlock one of threads waiting on the condition variable.
2 void
3 cond_signal (condvar_t *cvp) {
4     //LAB7 EXERCISE1: YOUR CODE
5     cprintf("cond_signal begin: cvp %x, cvp->count %d, cvp->owner->next_count
6 %d\n", cvp, cvp->count, cvp->owner->next_count);
7     /*
8      *      cond_signal(cvp) {
9      *          if(cv.count>0) {
10         *              mt.next_count++;
11         *              signal(cv.sem);
12         *              wait(mt.next);
13         *              mt.next_count--;
14         *      }
15     */
16     // 如果有进程在等待
17     if(cvp->count > 0) {
18         // 增加next等待计数
19         cvp->owner->next_count++;
20         // 唤醒一个等待进程
21         up(&(cvp->sem));
22         // 自身在next上等待（让出管程）
23         down(&(cvp->owner->next));
24         // 被唤醒后减少计数
25         cvp->owner->next_count--;
26     }
27     cprintf("cond_signal end: cvp %x, cvp->count %d, cvp->owner->next_count
28 %d\n", cvp, cvp->count, cvp->owner->next_count);
29 }
30
31 // Suspend calling thread on a condition variable waiting for condition
32 // Atomically unlocks
33 // mutex and suspends calling thread on conditional variable after waking up
34 // locks mutex. Notice: mp is mutex semaphore for monitor's procedures
35 void
36 cond_wait (condvar_t *cvp) {
37     //LAB7 EXERCISE1: YOUR CODE
38     cprintf("cond_wait begin: cvp %x, cvp->count %d, cvp->owner->next_count
39 %d\n", cvp, cvp->count, cvp->owner->next_count);
40     /*
41      *      cv.count++;
42      *      if(mt.next_count>0)
43      *          signal(mt.next)
44      *      else
45      *          signal(mt.mutex);
46      *      wait(cv.sem);
47      *      cv.count--;
48     */
49     // 增加等待计数
50     cvp->count++;
51     // 如果有进程在next上等待
52     if(cvp->owner->next_count > 0)
```

```
49     // 唤醒其中一个
50     up(&(cvp->owner->next));
51 // 否则释放管程锁
52 else
53     up(&(cvp->owner->mutex));
54 // 在条件变量上等待
55 down(&(cvp->sem));
56 // 被唤醒后减少计数
57 cvp->count--;
58 cprintf("cond_wait end:  cvp %x,  cvp->count %d,  cvp->owner->next_count
59 %d\n",  cvp,  cvp->count,  cvp->owner->next_count);
{}
```

使用`make grade`命令可以得到满分，所有输出检测都显示`OK`，如下所示：

```
tom@ubuntu:~/Downloads/riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-lin
ux-ubuntu14/labcode/labcode/lab7$ make grade
matrix:          (4.4s)
-check result:          OK
-check output:          OK
Total Score: 50/50
```

使用`make qemu`命令运行结果如下所示，输出了很多内容，包含五个哲学家就餐和思考的过程。


```
pid 12 done!.
pid 13 is running (13100 times)!.
pid 14 is running (23500 times)!.
pid 15 is running (41000 times)!.
pid 16 is running (3500 times)!.
pid 16 done!.
pid 17 is running (15400 times)!.
pid 18 is running (41000 times)!.
pid 19 is running (5900 times)!.
pid 19 done!.
pid 20 is running (26600 times)!.
pid 21 is running (3500 times)!.
pid 21 done!.
pid 22 is running (26600 times)!.
pid 23 is running (5900 times)!.
pid 23 done!.
pid 24 is running (37100 times)!.
pid 25 is running (15400 times)!.
pid 26 is running (3500 times)!.
pid 26 done!.
pid 27 is running (33400 times)!.
pid 28 is running (17900 times)!.
Iter 1, No.0 philosopher_sema is eating
Iter 1, No.2 philosopher_sema is eating
pid 13 done!.
pid 14 done!.
pid 17 done!.
pid 20 done!.
pid 25 done!.
pid 28 done!.
Iter 2, No.0 philosopher_sema is thinking
Iter 2, No.2 philosopher_sema is thinking
pid 22 done!.
pid 24 done!.
pid 27 done!.
pid 15 done!.
Iter 1, No.4 philosopher_sema is eating
Iter 1, No.1 philosopher_sema is eating
pid 18 done!.
matrix pass.
Iter 2, No.4 philosopher_sema is thinking
Iter 1, No.3 philosopher_sema is eating
Iter 2, No.1 philosopher_sema is thinking
Iter 2, No.0 philosopher_sema is eating
Iter 2, No.3 philosopher_sema is thinking
Iter 2, No.2 philosopher_sema is eating
Iter 3, No.0 philosopher_sema is thinking
Iter 2, No.4 philosopher_sema is eating
Iter 3, No.2 philosopher_sema is thinking
Iter 2, No.1 philosopher_sema is eating
Iter 3, No.4 philosopher_sema is thinking
Iter 2, No.3 philosopher_sema is eating
Iter 3, No.1 philosopher_sema is thinking
Iter 3, No.0 philosopher_sema is eating
Iter 3, No.3 philosopher_sema is thinking
Iter 3, No.2 philosopher_sema is eating
Iter 4, No.0 philosopher_sema is thinking
Iter 3, No.4 philosopher_sema is eating
```

```
Iter 4, No.2 philosopher_sema is thinking
Iter 3, No.1 philosopher_sema is eating
Iter 4, No.4 philosopher_sema is thinking
Iter 3, No.3 philosopher_sema is eating
Iter 4, No.1 philosopher_sema is thinking
Iter 4, No.0 philosopher_sema is eating
Iter 4, No.3 philosopher_sema is thinking
Iter 4, No.2 philosopher_sema is eating
No.0 philosopher_sema quit
Iter 4, No.4 philosopher_sema is eating
No.2 philosopher_sema quit
Iter 4, No.1 philosopher_sema is eating
No.4 philosopher_sema quit
```

```
Iter 4, No.3 philosopher_sema is eating
No.1 philosopher_sema quit
No.3 philosopher_sema quit
I am No.0 philosopher_condvar
Iter 1, No.0 philosopher_condvar is thinking
I am No.1 philosopher_condvar
Iter 1, No.1 philosopher_condvar is thinking
I am No.2 philosopher_condvar
Iter 1, No.2 philosopher_condvar is thinking
I am No.3 philosopher_condvar
Iter 1, No.3 philosopher_condvar is thinking
I am No.4 philosopher_condvar
Iter 1, No.4 philosopher_condvar is thinking
phi_test_condvar: state_condvar[0] will eating
phi_test_condvar: signal self_cv[0]
cond_signal begin: cvp c055a010, cvp->count 0, cvp->owner->next_count 0
cond_signal end: cvp c055a010, cvp->count 0, cvp->owner->next_count 0
Iter 1, No.0 philosopher_condvar is eating
cond_wait begin: cvp c055a038, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[2] will eating
phi_test_condvar: signal self_cv[2]
cond_signal begin: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
cond_signal end: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
Iter 1, No.2 philosopher_condvar is eating
cond_wait begin: cvp c055a088, cvp->count 0, cvp->owner->next_count 0
cond_wait begin: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[4] will eating
phi_test_condvar: signal self_cv[4]
cond_signal begin: cvp c055a0b0, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 1
Iter 1, No.4 philosopher_condvar is eating
cond_signal end: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 0
Iter 2, No.0 philosopher_condvar is thinking
phi_test_condvar: state_condvar[1] will eating
phi_test_condvar: signal self_cv[1]
cond_signal begin: cvp c055a038, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a038, cvp->count 0, cvp->owner->next_count 1
Iter 1, No.1 philosopher_condvar is eating
cond_signal end: cvp c055a038, cvp->count 0, cvp->owner->next_count 0
Iter 2, No.2 philosopher_condvar is thinking
phi_test_condvar: state_condvar[3] will eating
phi_test_condvar: signal self_cv[3]
cond_signal begin: cvp c055a088, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a088, cvp->count 0, cvp->owner->next_count 1
Iter 1, No.3 philosopher_condvar is eating
cond_signal end: cvp c055a088, cvp->count 0, cvp->owner->next_count 0
Iter 2, No.4 philosopher_condvar is thinking
cond_wait begin: cvp c055a010, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[0] will eating
phi_test_condvar: signal self_cv[0]
cond_signal begin: cvp c055a010, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a010, cvp->count 0, cvp->owner->next_count 1
Iter 2, No.0 philosopher_condvar is eating
cond_signal end: cvp c055a010, cvp->count 0, cvp->owner->next_count 0
Iter 2, No.1 philosopher_condvar is thinking
cond_wait begin: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[2] will eating
phi_test_condvar: signal self_cv[2]
cond_signal begin: cvp c055a060, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a060, cvp->count 0, cvp->owner->next_count 1
Iter 2, No.2 philosopher_condvar is eating
cond_signal end: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
Iter 2, No.3 philosopher_condvar is thinking
cond_wait begin: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 0
```



```
phi_test_condvar: state_condvar[0] will eating
phi_test_condvar: signal self_cv[0]
cond_signal begin: cvp c055a010, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a010, cvp->count 0, cvp->owner->next_count 1
Iter 4, No.0 philosopher_condvar is eating
cond_signal end: cvp c055a010, cvp->count 0, cvp->owner->next_count 0

Iter 4, No.1 philosopher_condvar is thinking
cond_wait begin: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[2] will eating
phi_test_condvar: signal self_cv[2]
cond_signal begin: cvp c055a060, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a060, cvp->count 0, cvp->owner->next_count 1
Iter 4, No.2 philosopher_condvar is eating
cond_signal end: cvp c055a060, cvp->count 0, cvp->owner->next_count 0
Iter 4, No.3 philosopher_condvar is thinking
cond_wait begin: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[4] will eating
phi_test_condvar: signal self_cv[4]
cond_signal begin: cvp c055a0b0, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 1
Iter 4, No.4 philosopher_condvar is eating
cond_signal end: cvp c055a0b0, cvp->count 0, cvp->owner->next_count 0
No.0 philosopher_condvar quit
cond_wait begin: cvp c055a038, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[1] will eating
phi_test_condvar: signal self_cv[1]
cond_signal begin: cvp c055a038, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a038, cvp->count 0, cvp->owner->next_count 1
Iter 4, No.1 philosopher_condvar is eating
cond_signal end: cvp c055a038, cvp->count 0, cvp->owner->next_count 0
No.2 philosopher_condvar quit
cond_wait begin: cvp c055a088, cvp->count 0, cvp->owner->next_count 0
phi_test_condvar: state_condvar[3] will eating
phi_test_condvar: signal self_cv[3]
cond_signal begin: cvp c055a088, cvp->count 1, cvp->owner->next_count 0
cond_wait end: cvp c055a088, cvp->count 0, cvp->owner->next_count 1
Iter 4, No.3 philosopher_condvar is eating
cond_signal end: cvp c055a088, cvp->count 0, cvp->owner->next_count 0
No.4 philosopher_condvar quit
No.1 philosopher_condvar quit
No.3 philosopher_condvar quit
all user-mode processes have quit.
init check memory pass.
kernel panic at kern/process/proc.c:536:
    initproc exit.
```