Week12 Report in Class (Fri56)

11911839 聂雨荷

Q1

解释 `local_intr_save(intr_flag);`的作用

因为内核在执行的过程中可能会被外部的中断打断，我们实现的ucore是不可抢占的系统，所以操作系统进行某些同步互斥的操作的时候需要先禁用中断，等执行完之后再打开中断

这里 `local_intr_save` 的作用是设置 sstatus 寄存器的 SIE 位为 1，表示开启中断。

```
#define local_intr_save(x)      do { x = __intr_save(); } while (0)

static inline bool __intr_save(void) {
    if (read_csr(sstatus) & SSTATUS_SIE) {
        intr_disable();
        return 1; // SIE = 1
    }
    return 0;
}

/* intr_disable - disable irq interrupt */
void intr_disable(void) { clear_csr(sstatus, SSTATUS_SIE); }
```

Q2

在`proc.c`中，`init_main`在852行执行了`check_sync()`方法。方法通过`sync/check_sync.c`中part1的算法解决哲学家吃饭问题。

（1）请描述part1的算法，并回答该算法是否能避免死锁？为什么？

（2）注释掉part1，并在part2中实现理论课件中哲学家问题的final solution算法（代码截图，运行结果截图）

(1) part1 算法如下，它可以避免死锁问题。

```
void phi_take_forks_sema(int i)
{
        down(&mutex);
        down(&s[i]);
        down(&s[RIGHT]);
        up(&mutex);
}

void phi_put_forks_sema(int i)
{
        up(&s[RIGHT]);
        up(&s[i]);
}
```

在这里，`mutex` semaphore 被初始化设置成了 1。由于它的存在，最多只有一个哲学家是 hold and wait 的状态，即只有一个人被阻塞。哲学家问题中，死锁来源于 5 个哲学家都在阻塞状态，而由于 semaphore 的存在，使得这样的问题不可能发生。故 part1 算法可以避免死锁。

(2) part2 - final solution

```
void phi_test_sema(int i)
{
    if(state_sema[i] == HUNGRY && state_sema[LEFT] !=
EATING && state_sema[RIGHT] != EATING){
        state_sema[i] = EATING;
        up(&s[i]);
    }
}

void phi_take_forks_sema(int i)
{
    down(&mutex);
    state_sema[i] = HUNGRY;
    phi_test_sema(i);
    up(&mutex);
    down(&s[i]);
}

void phi_put_forks_sema(int i)
{
    down(&mutex);
    state_sema[i] = THINKING;
    phi_test_sema(LEFT);
    phi_test_sema(RIGHT);
```

```
    up(&mutex);
}
```

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

```
No.1 philosopher_sema quit
No.3 philosopher_sema quit
Iter 1, No.4 philosopher_sema is eating
Iter 2, No.4 philosopher_sema is thinking
Iter 2, No.4 philosopher_sema is eating
Iter 3, No.4 philosopher_sema is thinking
Iter 3, No.4 philosopher_sema is eating
Iter 4, No.4 philosopher_sema is thinking
Iter 4, No.4 philosopher_sema is eating
No.4 philosopher_sema quit
all user-mode processes have quit.
init check memory pass.
kernel panic at kern/process/proc.c:464:
    initproc exit.
```