1. local_intr_save(intr_flag); `local_intr_save` 宏会将 `intr_flag` 的值设置为当前中断状态，并将中断禁止。
2. 这个算法采用了信号量来实现多进程之间的同步，哲学家先尝试获取互斥锁，然后通过修改信号量来使得自己成功获取到叉子。如果哲学家没有获取到叉子，由于其不会释放锁，所以会等待其他人释放叉子之后才会继续下去，其他人也会在获得锁之后尝试获取叉子，因此不会造成死锁。
3.

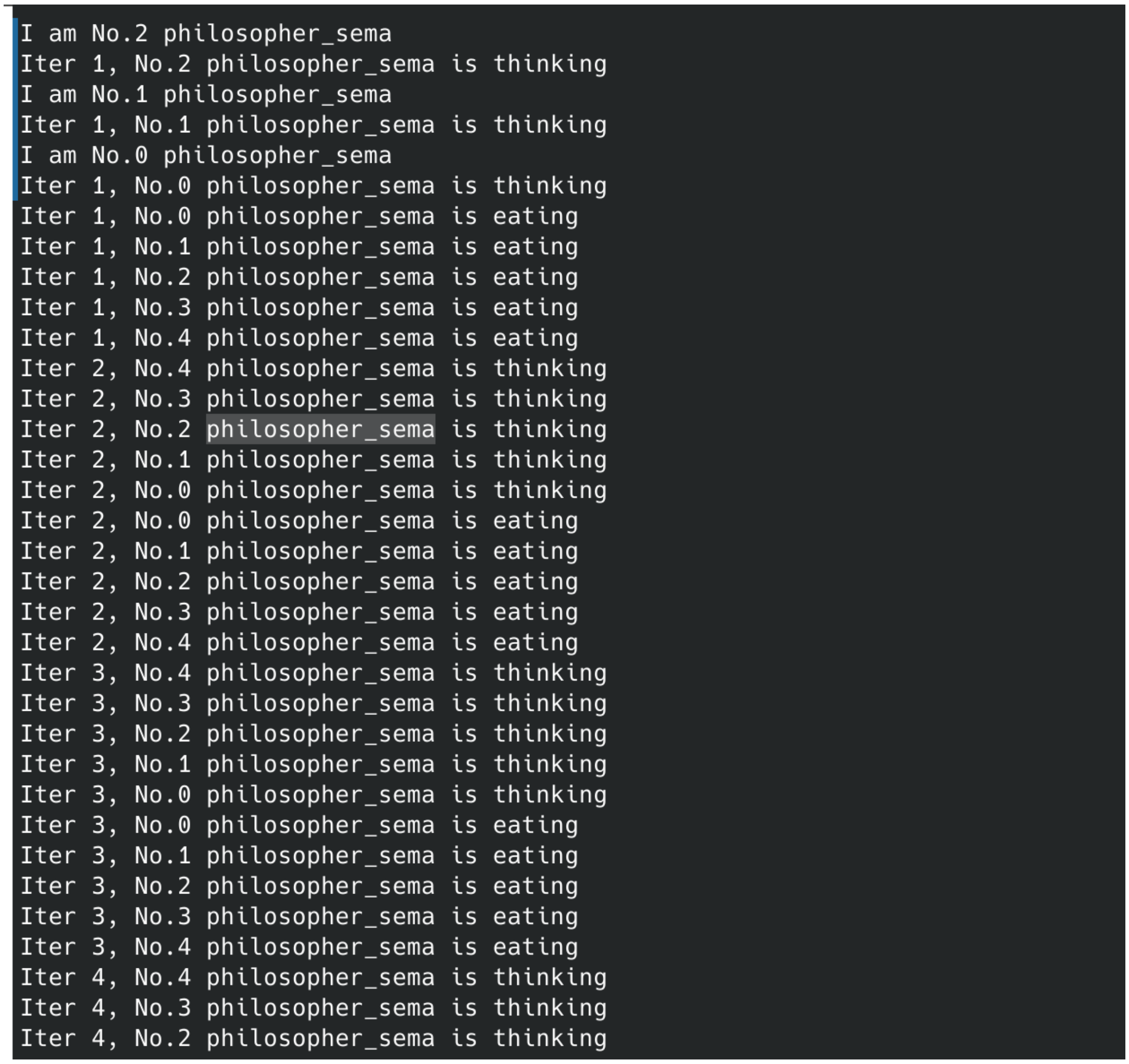
```c
#define sem_wait down
#define sem_post up
#define sem_signal up
//---------- philosophers problem using semaphore ----------------------
int state_sema[N]; /* 记录每个人状态的数组 */
int request[N]; /* 记录每个人是否需要叉子 */
semaphore_t mutex;
semaphore_t s[N];

struct proc_struct *philosopher_proc_sema[N];

//---------------part2-----------------
void phi_test_sema(int i) {
    if (state_sema[i] == 1 && !request[(i + 1) % N]) {
        state_sema[i] = 2;
        sem_signal(&s[i]);
    } else if (state_sema[i] == 1 && !request[(i + N - 1) % N]) {
        state_sema[i] = 2;
        sem_signal(&s[(i + N - 1) % N]);
    }
}

void phi_take_forks_sema(int i) {
    sem_wait(&mutex);
    state_sema[i] = 1;
    phi_test_sema(i);
    sem_post(&mutex);
    sem_wait(&s[i]);
}

void phi_put_forks_sema(int i) {
    sem_wait(&mutex);
    state_sema[i] = 0;
    phi_test_sema((i + N - 1) % N);
    phi_test_sema((i + 1) % N);
    sem_post(&mutex);
}
```



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