实验五报告

实现的功能

- 将相关数组与判断方法集中到 `struct DeadlockDetect` 中
- 分别实现了互斥锁和信号量的银行家算法

问答题

1. 在我们的多线程实现中，当主线程（即 0 号线程）退出时，视为整个进程退出， 此时需要结束该进程管理的所有线程并回收其资源。

   - 需要回收的资源有哪些?
   - 其他线程的 `TaskControlBlock` 可能在哪些位置被引用，分别是否需要回收，为什么?

   对比以下两种 `Mutex.unlock` 的实现，二者有什么区别? 这些区别可能会导致什么问题?

```
  impl Mutex for Mutex1 {
    fn unlock(&self) {
        let mut mutex_inner = self.inner.exclusive_access();
        assert!(mutex_inner.locked);
        mutex_inner.locked = false;
        if let Some(waking_task) = mutex_inner.wait_queue.pop_front() {
            add_task(waking_task);
        }
    }
}

impl Mutex for Mutex2 {
    fn unlock(&self) {
        let mut mutex_inner = self.inner.exclusive_access();
        assert!(mutex_inner.locked);
        if let Some(waking_task) = mutex_inner.wait_queue.pop_front() {
            add_task(waking_task);
        } else {
          mutex_inner.locked = false;
        }
    }
}
```

   区别在于 Mutex1 先释放锁再唤醒线程，Mutex2 先唤醒再释放锁
   Mutex1 无论是否有等待的线程，都立即释放锁
   Mutex2 如果没有等待欲线程，将不会再释放锁，减少操作

荣誉准则

1. 在完成本次实验的过程（含此前学习的过程）中，我曾分别与 以下各位 就（与本次实验相关的）以下方面做过交流，还在代码中对应的位置以注释形式记录了具体的交流对象及内容：

   无

2. 此外，我也参考了 以下资料 ，还在代码中对应的位置以注释形式记录了具体的参考来源及内容：

   无

3. 我独立完成了本次实验除以上方面之外的所有工作，包括代码与文档。 我清楚地知道，从以上方面获得的信息在一定程度上降低了实验难度，可能会影响起评分。

4. 我从未使用过他人的代码，不管是原封不动地复制，还是经过了某些等价转换。 我未曾也不会向他人（含此后各届同学）复制或公开我的实验代码，我有义务妥善保管好它们。 我提交至本实验的评测系统的代码，均无意于破坏或妨碍任何计算机系统的正常运转。 我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按“-100”分计。