计算机学院_操作系统_课程实验报告

实验题目： 死锁问题		学号：202200130048
日期：11.13	班级： 6	姓名： 陈静雯
Email：1205037094@qq.com		

实验步骤与现象：

1. 示例实验

```
orange@orange-VirtualBox:~/czsystem/test6/exam$ ./dp 1
orange@orange-VirtualBox:~/czsystem/test6/exam$ p1:2964 hungry
p2:2965 eating
p3:2966 hungry
p4:2967 eating
p5:2968 hungry
p2:2965 thinking
p1:2964 eating
p4:2967 thinking
p2:2965 hungry
p1:2964 thinking
p2:2965 eating
p4:2967 eating
p2:2965 thinking
p1:2964 eating
p4:2967 thinking
p1:2964 thinking
p5:2968 eating
p2:2965 hungry
p4:2967 hungry
p5:2968 thinking
p1:2964 eating
p5:2968 hungry
p1:2964 thinking
```

可以看到 5 个哲学家进程在 3 中状态中不断的轮流变换，且连续的 5 个输出中不多于 2 个的状态为 eating。

（1）死锁

每个哲学家首先会试图获取右边的筷子，然后才是左边的筷子。如果所有哲学家几乎同时开始这个过程，并且每个哲学家都恰好在另一个哲学家之后获得了右边的筷子，那么每个哲学家都将持有右边的筷子并等待左边的筷子，形成死锁。

```
void dp::pickup(int i) {
    lock->close_lock();
    *state[i] = hungry;
    // 先尝试获取右边的筷子
    self[(i + 1) % 5]->Wait(lock, (i + 1) % 5);
    // 再尝试获取左边的筷子
    self[i]->Wait(lock, i);
    cout << "p" << i + 1 << ":" << getpid() << " eating\n";
    sleep(rate);
    lock->open_lock();
}
```

（2）饥饿

在这个例子中，哲学家 1 总是最后一个尝试获取筷子，因此在高竞争的情况下，他可能会因为其他哲学家总是更快地获取筷子而长时间处于饥饿状态。这种饥饿现象是因为哲学家 1 没有机会在其他哲学家之前获取筷子，尤其是在其他哲学家频繁交替使用筷子的情况下。

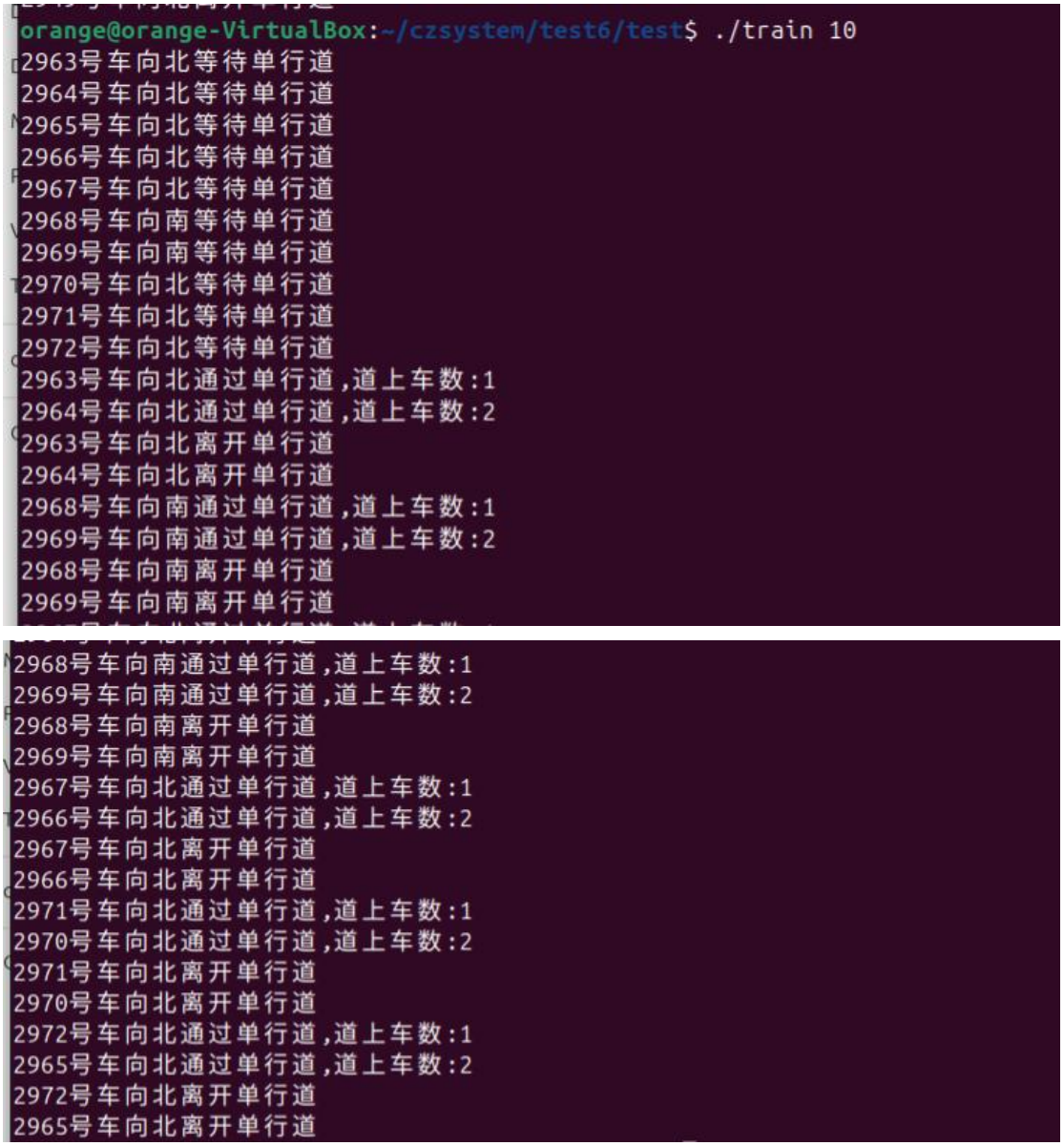
```
// 最后创建哲学家1的进程
pid[0] = fork();
if (pid[0] < 0) {
    perror("p1 create error");
    exit(EXIT_FAILURE);
} else if (pid[0] == 0) {
    while (1) {
        tdp->pickup(0);
        tdp->putdown(0);
    }
}

return 0;
```

2. 独立实验

多列火车可以分别从两个城市的车站，排队等待进入车道向对方城市行驶，该铁路在同一时间，只能允许在同一方向上行车，如果同时有相向的火车行驶将会撞车。请模拟实现两个方向行车，而不会出现撞车或长时间等待的情况。

默认参数是五辆车，可以输入参数；一条道最多两辆车（默认）

```
orange@orange-VirtualBox:~/czsystem/test6/test$ ./train
2947号车向北等待单行道
2948号车向北等待单行道
2949号车向北等待单行道
2950号车向北等待单行道
2951号车向南等待单行道
2947号车向北通过单行道,道上车数:1
2948号车向北通过单行道,道上车数:2
2947号车向北离开单行道
2948号车向北离开单行道
2951号车向南通过单行道,道上车数:1
2951号车向南离开单行道
2950号车向北通过单行道,道上车数:1
2949号车向北通过单行道,道上车数:2
2950号车向北离开单行道
2949号车向北离开单行道
```


orange@orange-VirtualBox:~/czsystem/test6/test$ ./train 10
2963号车向北等待单行道
2964号车向北等待单行道
2965号车向北等待单行道
2966号车向北等待单行道
2967号车向北等待单行道
2968号车向南等待单行道
2969号车向南等待单行道
2970号车向北等待单行道
2971号车向北等待单行道
2972号车向北等待单行道
2963号车向北通过单行道,道上车数:1
2964号车向北通过单行道,道上车数:2
2963号车向北离开单行道
2964号车向北离开单行道
2968号车向南通过单行道,道上车数:1
2969号车向南通过单行道,道上车数:2
2968号车向南离开单行道
2969号车向南离开单行道

2968号车向南通过单行道,道上车数:1
2969号车向南通过单行道,道上车数:2
2968号车向南离开单行道
2969号车向南离开单行道
2967号车向北通过单行道,道上车数:1
2966号车向北通过单行道,道上车数:2
2967号车向北离开单行道
2966号车向北离开单行道
2971号车向北通过单行道,道上车数:1
2970号车向北通过单行道,道上车数:2
2971号车向北离开单行道
2970号车向北离开单行道
2972号车向北通过单行道,道上车数:1
2965号车向北通过单行道,道上车数:2
2972号车向北离开单行道
2965号车向北离开单行道

结论分析：

1. 示例实验是否真正模拟了哲学家就餐问题？

是。通过使用信号量和条件变量来模拟哲学家的就餐过程。每个哲学家在尝试就餐之前都会检查左右两边的哲学家是否正在就餐，如果两边都没有在就餐，那么该哲学家就可以开始就餐。

2. 为什么示例程序不会产生死锁？

（1） 资源分配策略：每个哲学家在尝试获取筷子（即就餐）之前会先检查左右两边的哲学家是否正在就餐。如果两边的哲学家中任一正在进行就餐，当前哲学家就会等待，直到左右两边的哲学家都不在就餐时才会开始就餐。这种策略避免了所有哲学家同时持有筷子并等待另一个筷子的情况，从而防止了死锁的发生。

（2） 使用信号量控制资源访问：程序中使用了一个全局信号量来控制对筷子资源的访问。当一个哲学家试图就餐时，他首先需要获得这个信号量，这保证了在同一时刻只有一个哲学家可以尝试获取筷子，进一步减少了死锁的可能性。

（3） 条件变量：每个哲学家都有一个与之对应的条件变量。当一个哲学家不能立即就餐时，他会释放锁并等待条件变量的通知。一旦左右两边的哲学家中有一个结束了就餐，就会唤醒相应的条件变量，使得等待中的哲学家有机会再次尝试就餐。

（4） 初始化状态：所有哲学家的初始状态都是思考（thinking），这意味着没有哲学家一开始就在尝试就餐，因此避免了初始状态下的死锁。

3. 为什么会出现进程死锁和饥饿现象？

死锁：

（1）互斥条件：资源不能被共享，只能被一个进程独占使用。

（2）占有且等待条件：进程已经占有某些资源，同时又申请其他资源。

（3）不可抢占条件：已分配的资源不能被强制回收，只能等待占用它们的进程自行释放。

（4）循环等待条件：存在一个进程等待环，每个进程都在等待下一个进程中所持有的资源。

饥饿：

饥饿是指一个或多个进程长期得不到所需的资源而无法运行。主要原因是资源分配策略不当，例如优先级反转或资源分配算法不公平等。

4. 怎样利用实验造成和表现死锁和饥饿现象？

死锁实现：使用线程和锁来模拟资源请求，故意让部分线程按相反顺序请求资源，观察程序是否进入死锁状态。

饥饿实现：故意使某些进程永远处于等待状态。

5. 管程能避免死锁和饥饿的机理是什么？

避免死锁

（1）互斥锁：确保每次只有一个进程可以进入临界区，避免多个进程同时请求资源。

（2）条件变量：允许进程在无法立即获取所需资源时等待，直到条件满足再继续执行，避免了循环等待。

（3）资源分配策略：通过合理的资源分配策略，确保不会形成资源请求环。

避免饥饿

（1）公平性：确保所有进程都有机会获得所需的资源，避免某些进程长期得不到服务。
（2）优先级管理：合理管理进程优先级，避免优先级反转导致的饥饿。

6. 对于管程概念有哪些新的理解和认识？

（1）封装性：管程将数据和操作封装在一起，提供了一个清晰的接口，使得并发编程更加模块化和易于理解。
（2）同步机制：管程内置了锁和条件变量，提供了强大的同步机制，使得开发者可以更方便地实现复杂的并发控制。
（3）安全性：管程通过互斥锁确保每次只有一个进程可以访问临界区，避免了数据竞争和不一致的问题。

7. 条件变量和信号量有何不同？

（1）信号量：主要用于控制对共享资源的访问，可以表示资源的数量。
提供 P（减一操作）和 V（加一操作），用于同步进程。可以表示资源的数量，适用于多种同步场景。
（2）条件变量：主要用于进程间的条件等待和通知，常用于更复杂的同步场景。
提供 wait 和 signal 操作，允许进程在某个条件不满足时等待，直到条件满足后再继续执行。与互斥锁结合使用，确保等待和通知操作的原子性。

8. 为什么在管程中要使用条件变量而不直接使用信号量来达到进程同步的目的？

（1）精确控制：条件变量允许进程在特定条件下等待和唤醒，提供了更细粒度的控制。
（2）避免忙等待：使用条件变量可以让进程在条件不满足时主动进入等待状态，而不是不断轮询，节省了 CPU 资源。
（3）组合灵活性：条件变量与互斥锁结合使用，可以更好地管理临界区和同步条件，避免了信号量可能引起的复杂性和错误。

9. 示例实验中构造的管程中的条件变量是一种什么样式的？

每个哲学家都有一个条件变量，用于表示其是否可以开始就餐。
（1）初始化：每个哲学家的条件变量在初始化时都设置为饥饿状态。
（2）等待：当哲学家不能立即就餐时，会调用 Wait 方法，释放锁并进入等待状态。
（3）唤醒：当左右邻居的哲学家结束就餐后，会调用 Signal 方法，唤醒等待中的哲学家。

```
for (int i = 0; i < 5; i++) {
    //为每个哲学家建立一个条件变量和可共享的状态 //初始状态都为思考
    if ((state[i] = (char *)set_shm(shm_key++, shm_num, ipc_flg)) =
        perror("Share memory create error");
        exit(EXIT_FAILURE);
    }
    *state[i] = thinking;
    //为每个哲学家建立初值为 0 的用于条件变量的信号量
    if ((sem_id = set_sem(sem_key++, sem_val, ipc_flg)) < 0) {
        perror("Semaphor create error");
        exit(EXIT_FAILURE);
    }
    sema = new Sema(sem_id);
```

10. 其中的锁起了什么样的作用？

（1）互斥：确保每次只有一个哲学家可以进入临界区，避免多个哲学家同时修改共享资源（如状态变量）。

（2）同步：配合条件变量使用，确保等待和唤醒操作的原子性，避免数据竞争和不一致的问题。

11. 独立实验程序是怎样解决单行道问题的？怎样构造管程对象的？

（1）管程对象的构造

a. 共享内存：用于存储当前方向、是否为第一辆车、反向车辆数量、当前通过的车辆数和当前方向等待的车辆数量。

b. 信号量：用于控制对共享资源的访问，确保互斥。

c. 条件变量：用于管理车辆的等待和唤醒操作。

```
dp::dp(int max, int cars)
{
    int ipc_flg = IPC_CREAT | 0644;
    int shm_key = 220;
    int shm_num = cars;
    int sem_key = 120;
    int sem_val = 0;
    int shm_flg = IPC_CREAT|0644;
    int sem_id;
    Sema *sema;
    Sema *sema1;
    Sema *sema2;
    maxCars = max;
    numCars = 0;

    //当前的方向
    curDire = (int *) set_shm(shm_key++, 1, shm_flg);
    //是否为第一辆车
    isFirst = (int *) set_shm(shm_key++, 1, shm_flg);
    //反向车辆的数量
```

（2）车辆到达(Ready 方法)

a. 检查是否为第一辆车：如果是第一辆车，设置当前方向。

b. 记录方向：记录车辆的方向并打印等待信息。

c. 检查当前方向：如果当前方向与车辆方向不同，增加反向车辆计数并等待。

```
void dp::Ready(int direc)
{
    int count;

    lock->close_lock();
    mutex->down();
    //获取是否为第一辆车
    count = *isFirst;
    mutex->up();

    //如果值为0说明是第一辆车
    if(count == 0)
    {
        mutex->down();
        *curDire = direc;
        *isFirst = ++count;
        mutex->up();

    }
```

（3）车辆通过（Cross 方法）

a.检查当前通过的车辆数：如果超过最大容量，增加当前方向等待的车辆计数并等待。

b.增加当前通过的车辆数：记录车辆通过并打印信息。

```
void dp::Cross(int direc)
{
    int count, i;
    lock->close_lock();

      //获得当前正在通过的车辆数
       numCars = getNum();
       if(numCars >= maxCars)
       {
           mutex->down();
           count = *curnum;
           *curnum = ++count;
           mutex->up();
           do{
               con->Wait(lock, direc);
               numCars = getNum();
           }while(numCars >= maxCars);
      }

    addNum();
    numCars = getNum();
```

（4）车辆离开（Leave 方法）

a.减少当前通过的车辆数：记录车辆离开并打印信息。

b.唤醒等待的车辆：如果当前没有车辆通过且有反向车辆等待，优先唤醒反向车辆；如果没有反向车辆等待，唤醒同方向的车辆。

```
void dp::Leave(int direc)
{
    int count, num;
    lock->close_lock();
    if(direc==0)
    {
        printf("%d号车向南离开单行道\n", getpid());
    }else{
    printf("%d号车向北离开单行道\n", getpid());
    }
    //减少当前车的数量
    decNum();
```