山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号：202100130022	姓名：郭家宁	班级：21 数据
实验题目：死锁问题		
实验学时：2	实验日期：2023-5-22	
实验目的：通过本实验观察死锁产生的现象，考虑解决死锁问题的方法。从而进一步加深对于死锁问题的理解。掌握解决死锁问题的几种算法的编程和调技术。练习怎样构造管程和条件变量，利用管程机制来避免死锁和饥饿问题的发生。		

实验结果：

一、 实例实验

1. 运行结果：

可以发现每个哲学家都均匀顺序的吃上了饭，没有出现死锁和一个哲学加一直饥饿的现象。

```
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_example$ ./dp
p1:6960 eating
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_example$ p4:6963 eating
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_example$ p5:6964 hungry
p3:6962 hungry
p2:6961 hungry
p1:6960 thinking
p4:6963 thinking
p5:6964 eating
p2:6961 eating
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_example$ p1:6960 hungry
p4:6963 hungry
p5:6964 thinking
p2:6961 thinking
p4:6963 eating
p1:6960 eating
p5:6964 hungry
p2:6961 hungry
p1:6960 thinking
p4:6963 thinking
p5:6964 eating
p3:6962 eating
p1:6960 hungry
p4:6963 hungry
p5:6964 thinking
p3:6962 thinking
p1:6960 eating
p4:6963 eating
p1:6960 thinking
p5:6964 hungry
p3:6962 hungry
p4:6963 thinking
```

二、 独立实验

1. 在两个城市南北方向之间存在一条铁路，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。请构造一个

管程来解决该问题。

2. 构造的火车站的管程定义如下：

```
//火车站管程的定义
class dp
{
public:
    int *maxTrains ; //最大火车数
    int *nowTrains; //当前已经通过的火车数

    int *overNorth; //当前已经通过的由南向北的火车数
    int *overSouth; //当前已经通过的由北向南的火车数
    dp(int rate, int maxcur); //管程构造函数
    ~dp();
    void start(int i);
    void quit(int i);
    //建立或获取 ipc 信号量的一组函数的原型说明
    int get_ipc_id(char *proc_file, key_t key);
    int set_sem(key_t sem_key, int sem_val, int sem_flag);
    char *set_shm(key_t shm_key, int shm_num, int shm_flag);
private:
    int rate; //模拟发车的速度
    Lock *lock;
    char *state[2];     //两个火车站的状态
    int cnt[2];         //火车站同时发送火车的数量
    Condition *self[2]; //火车站条件变量
};
```

3. 使用两个子进程来模拟两个火车站，初始默认火车站的火车数量的和为 10，一次最大通行数量为 2，由南向北的车站火车为 4 辆，由北向南的火车为 6 辆。

4. 火车的发车函数：首先上锁，然后判断是否可以发车，不能发车就阻塞自己，若能发车，就将自己已经发车的数量加一。

5. 火车站的发车结束函数：上锁，如果已经到达一次性发车的最大数量，然后判断对面是否已经把所有的车发送完毕，如果没有，就把自己的状态设为等待中，而且唤醒另一个车站，对面如果把所有的车发送结束，

就继续发车直到自己发车结束，发车结束处理完后开锁。

```
lock->close_lock();
   if (i == 0)
   {
       if ((cnt[0] == MaxPass || *overNorth == northSum))
       {
           if (*overSouth < southSum)
           {
               cnt[0] = 0;
               *state[0] = waitt;
               self[1]->Signal(1); // 唤醒对面车站
               if (*overNorth < northSum)
                   cout << getpid() << "号火车站等待" << endl;
               else
                   cout << getpid() << "号火车站结束发车" << endl;
           }
           else if (*overNorth < northSum)
           {
               cout << getpid() << "号火车站继续发车" << endl;
           }
           else
           {
               cout << getpid() << "号火车站结束发车" << endl;
           }
       }
       else
       {
           cout << getpid() << "号火车站继续发车" << endl;
       }
   }
```

6. 运行结果如下：

```
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_test$ ./dp
6904号火车站发送0号火车由北向南
6904号火车站继续发车
6904号火车站发送1号火车由北向南
6904号火车站等待
6903号火车站发送2号火车由南向北
6903号火车站继续发车
6903号火车站发送3号火车由南向北
6903号火车站等待
6904号火车站发送4号火车由北向南
6904号火车站继续发车
6904号火车站发送5号火车由北向南
6904号火车站等待
6903号火车站发送6号火车由南向北
6903号火车站继续发车
6903号火车站发送7号火车由南向北
6903号火车站结束发车
6904号火车站发送8号火车由北向南
6904号火车站继续发车
6904号火车站发送9号火车由北向南
6904号火车站结束发车
jianing@jianing-virtual-machine:~/桌面/os_ex/exe6/exe6_test$
```

可以看出没有出现死锁问题，列车成功把所有的列车发送出去。

三、 实验要求

1. 实力实验使用管程成功实现了哲学家就餐问题，如果要发生死锁和饥饿。

死锁：

每位哲学家都先拿起自己左边的一只筷子，随后再分别拿右边的一只筷子，那么可能出现所有人都只拿到了左边或者右边的筷子而无法实现进一步的操作，从而导致死锁的发生。

饥饿：

即在开始的哲学家就餐结束后，不发送 Signal 唤醒，这时所有的哲学家都不在就餐，两个哲学家一直就餐。

2. 管程能避免死锁和饥饿的机理是什么？您对于管程概念有哪些新的理解和认识？条件变量和信号量有何不同？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） 管程是一种进程同步机制，可以用来管理共享资源的访问。它提供了一组互斥量和条件变量，可以避免死锁和饥饿，确保多个进程或线程可以安全地访问共享资源。

机理：

避免死锁：当一个进程请求访问共享资源时，如果该资源被锁定，则该进程会阻塞，并等待资源释放。在管程中，如果一个进程无法获得共享资源的锁定时，它会等待在进程队列中，直到资源可用为止，从而避免死锁情况的发生。

避免饥饿：在管程中，每个进程都有一个优先级，并按照一定规则分配 CPU 时间片，使得高优先级的进程更容易获得资源的锁定。同时，管程也提供了条件变量，可以让等待时间较长的进程优先获得资源的机会，从而避免饥饿。

（2） 对于管程概念的新认识：

管程提供了一种内部使用的同步机制，主要用于管理共享资源，可以避免死锁和饥饿的发生。

管程是一种高级抽象，可以封装了复杂的进程同步机制，使得并发编程变得更加简单和可读性好。

（3） 条件变量和信号量的区别以及为什么要在管程中使用条件变量：

条件变量是一种用于进程同步的工具，可以唤醒等待某个特定事件（条件）的其他进程。常见操作包括：wait、signal、broadcast 等。

信号量是另一种常用的进程同步机制，例如二进制信号量和计数器信号量，它们分别用于管理共享资源的互斥访问和有限资源的分配等问题。

在管程中，条件变量更适合用于阻塞和唤醒等待共享资源的进程，而信号量更适合用于控制访问共享资源的互斥操作。此外，管程提供的条件变量实现了一个全新的屏障概念，即当调用 wait 函数时当前线程会释放互斥锁，进入等待队列，直到相应条件被 Signal 唤醒以后。

问题及收获： 对死锁和饥饿问题和管程防止死锁的机制有了更加深刻的理解与体会。