深入理解管程

思路：

- 把PV操作中的信号量设置为条件变量，并对每一个条件变量设置一个计数器
- 设置判定边界所用的变量（管程不能通过信号量直接判断是否有资源，因此应该单独设置变量）

⚠️注意：管程和PV有个区别是：PV只需要根据需要设定计数器（因为信号量的数值本身可以记录进程数量），但管程需要为每个可并行运行的进程设置计数器。或者设置状态flag。

读者-写者问题 -- 写进程优先

```
// ppt中的答案
TYPE read-write = monitor{
    int rc = 0,wc = 0;//读者/写者数量
    semaphore R = 0,W = 0;
    int R_count , W_count//正在阻塞的读者/写者数量

    InterfaceModule IM;
    define start_read,end_read,start_write,ent_write;
    use wait,signal,enter,leave;
}
void start_write(){
    enter(IM);
    wc++;
    if(rc>0 || wc>1){ // ? wc > 1 是个什么鬼    wc>0?
        wait(W,W_count,IM)
    }
    leave(IM)
}
void end_write(){
    enter(IM)
    wc--;
    if(wc>0){
        signal(W,W_count,IM)
    }else{
        signal(R,R_count,IM)
```

```
    }
}
void start_read(){
  enter(IM)
  rc++
  if(wc>0){
    wait(R,R_count,IM)
  }
  leave(IM)
}
void end_read(){
  enter(IM)
  rc--;
  if(rc==0){
    signal(W,W_count,IM)
  }
}

process reader(){
  read-write.start_read()
  //read
  read-write.end_read();
}
process writer(){
  read-write.start_write();
  //write
  read-write.end_write();
}
```

```
type reader_writer = monitor{
    int rc = 0,wc = 0;//正在工作的读者/写者数目
    semaphore read,writer;
    int read_count,write_count;

    InterfaceModule IM;
    define start_read,end_read,start_write,end_write;
    use enter,leave,wait,signal;
}
void start_read(InterfaceModule &IM){
    enter(IM);
    if(wc>0){
        wait(reader,reader_count,IM);
    }
    rc++;
    leave(IM);
}
void end_read(InterfaceModule &IM){
    enter(IM);
    rc--;
```

```
    leave(IM)
}
void start_write(Interface & IM){
    enter(IM);
    if(wc>0)||rc>0{//有读者在读，或有读者在写，写者都不能写
        wait(write,write_count,IM);
    }
    wc++;
    leave(IM);
}
void end_write(Interface &IM){
    enter(IM);
    wc--;
    if(wc==0){
        signal(reader,reader_count,IM);
    }else{
        signal(writer,writer_count,IM);
    }
}
```

读者-写者问题 -- 读进程优先

```
type reader_writer = monitor{
    int rc = 0,wc = 0;
    semaphore reader,writer;
    int reader_count,writer_count;
    InterfaceModule IM;
    define start_read,end_read,start_write,end_write;
    use enter,leave,wait,signal;
}
void start_read(){
    enter(IM);
    if(wc>0){
        wait(read,read_count,IM);
    }
    rc++;
    leave();
}
void end_read(){
    enter();
    rc--;
    if(rc==0){
        signal(writer,writer_count,IM);
    }
    leave();
}
void start_write(){
    enter();
    if(rc>0||wc>0){
```

```
        wait(writer,writer_count,IM);
    }
    leave();
}
void end_write(){
    enter();
    wc--;
    leave();
}
```

生产者-消费者问题

```
TYPE producer-consumer = monitor{
  semaphore empty = k,full = 0;
  int empty_count = 0,full_count = 0 ;
  int count;//缓冲区的物品数
  int in,out;//存取指针
  InterfaceModule IM;
  define append,take;
  use enter,leave,wait, signal;
}
void append(item &x){
  enter(IM);
  if(count==k){
    wait(empty,empty_count,IM);
  }
    B[in] = x;
  in = (in + 1)%k;
  count++;
  signal(full,full_count,IM);
  leave(IM);
}
void take(item &x){
  enter(IM);
  if(count==0){
    wait(full,full_count,IM);
  }
  x = B[out];
  out = (out+1)%k;
  count--;
  singal(empty,empty_count,IM);
  leave(IM);
}
cobegin
process produce_i{
    item x;
    produce x;
```

```
    producer_consumer.append(x);
}
process consumer_i{
    item x;
    procuder_consumer.take(x);
    consume(x);
}
```

哲学家进餐问题

```
* 拿筷子:
        他饿了
        并且左右两边都有筷子，则拿起筷子
        如果不可以吃，则等待
* 放筷子:
        他吃完了
        放下筷子
        测试左右两边的邻居可不可以拿筷子

* 测试:
        如果左右两边不想拿筷子并且自己饿了
        则拿起筷子，吃饭

TYPE dining_philosophers = moniter{
  enum {thinking,hungry,eating} state[5];
  semaphore self[5] = thinking;
  int self_count[5] = {0};
  InterfaceModule IM;
  define pickup,putdown;
  use enter,leave,wait,signal;
}
void pickup(int i){
  enter(IM);
  state[i]=hungry;
  test(i);
  if(self[i]!=eating){
    wait(self[i],self_count[i],IM);
  }
  leave(IM);
}
void putdown(int i){
  enter(IM);
  state[i] = thinking;
  test((i-1)%5);
  test((i+1)%5);
  leave(IM);
}
void test(int k){
  if(state[(k-1)%]!=eating && state[k]==hungry && state[(k+1)%5]!=eating){
```

```
    state[k] = eating;
    signal(self[k],self_count[k],IM);
  }
}

cobegin
process philosopher_i(){
    while(true){
        thinking();
        dining_philosophers.pickup();
        eating();
        dining_philosophers.putdown();
    }
}
```

苹果-橘子问题

```
TYPE apple_orange = monitor{
  semaphore apple = 0,orange = 0;plate = 0;
  int apple_count = 0,orange_count = 0;plate_count = 0;
  bool full;
  InterfaceModule IM;
  define put , get;
  use enter,leave,wait,signal;
}
void put_apple(){
  if(full){
    wait(plate,plate_count,IM)
  }
  full = true;
  //放入水果
  signal(apple,apple_count,IM);
}
void put_orange(){
  if(full){
    wait(plate,plate_count,IM);
  }
  full = true;
  //放入水果
  signal(orange,orange_count,IM);
}
void get_apple(){
  if(!full){
    wait(apple,apple_count,IM);
  }
  //拿走苹果
  full = false;
  signal(plate,plate_count,IM);
```

```
}
void get_orange(){
  if(!full){
    wait(orange,orange_count,IM);
  }
  //拿走橘子
  full = false;
  signal(palte,plate_count,IM);
}
cobegin
process father(){
  //准备好苹果
  apple_orange.put_apple();
}
process mother(){
  //准备好橘子
  apple_orange.put_orange();
}
process son(){
  apple_orange.get_orange();
  //吃橘子
}
process daughter(){
  apple_orange.get_apple();
  //吃苹果
}
```

吸烟者问题

```
Type smokers = monitor{
    semaphore provider smoker1,smoker2,smoker3;
    int provider_count = 0, smoker1_count = 0,smoker2_count = 0,smoker3_count = 0;
    bool flag1 = flag2 = flag3 = true;
    InterfaceModule IM;
    define give,take1,take2,take3;
    use enter,leave,wait,signal;
}
void give(Interface IM){
    enter(&IM);
    if(flag1||flag2||flag3){
        wait(provider,provider_count,IM);
    }
    //准备原料
    if(原料2&&原料3){
        singal(smoker1,smoker1_count,IM);
    }else(原料1&&原料3){
        singal(smoker2,somker2_count,IM);
    }else{
        singal(smoker3,smoker3_count,IM);
```

```
    }
}
void take1(){
    if(!flag1){
        wait(smoker1,smoker1_count,IM);
    }
    //拿走
    flag1 = false;
    singal(provider,provider_count,IM);
}
cobegin{
    void provider(){
        while(true){
            smokers.give();
        }
    }
    void smoker1(){
        while(true){
            smokers.take1();
            //制作
            //吸烟
        }
    }
}
```