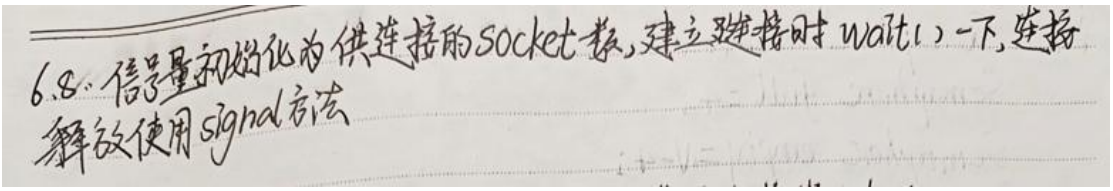
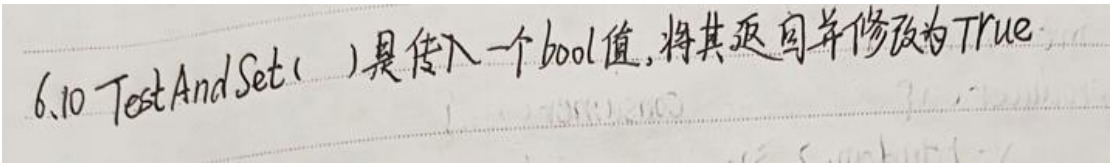
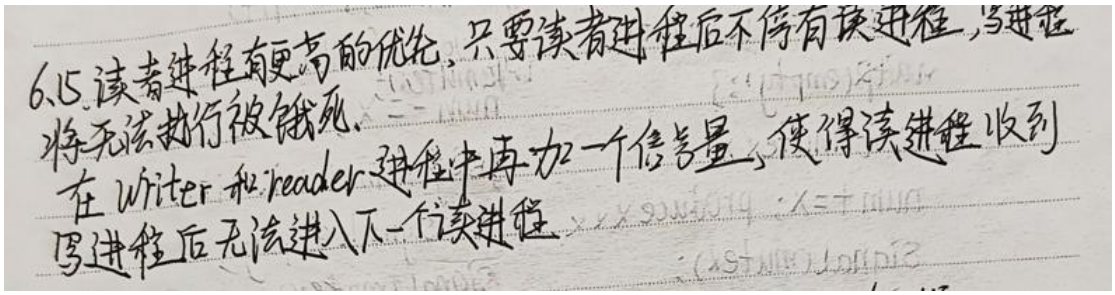
```
// 6.7
do
{
    // 初始化等待列表
    waiting[i] = TRUE;
    // 尝试获取锁
    key = TRUE;
    while (waiting[i] && key)
        key = Swap(&lock, &key);
    // 获取锁成功，开始执行临界区代码
    waiting[i] = FALSE;
    // 尝试获取下一个锁
    j = (i + 1) % n;
    while ((j != i) && !waiting[j])
        j = (j + 1) % n;
    // 如果获取成功，则设置锁为false
    if (j == i)
        lock = FALSE;
    else
        waiting[j] = FALSE;
    // 等待
} while (TRUE);
```

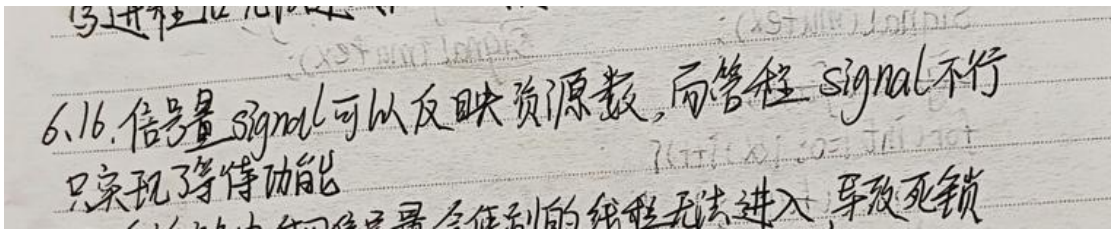


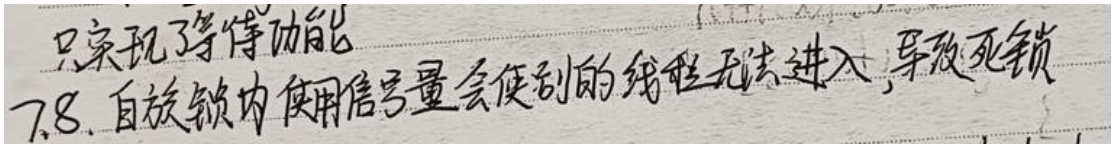


```
// 6.10
//互斥锁
int guard = 0;
int value = 0;
wait()
{
    while (TestAndSet(&guard) == 1);
    if (value == 0)//信号量为0
    {
        add process to the wait queue;
    }
    else
    {
        //取一个值
       value--;

       guard = 0;
    }
}
signal()
{
    while (TestAndSet(&guard) == 1);
    if (value == 0 && there is a process on the wait queue)
    wake up the first process on the wait queue;
    else
    {   //放一个值
        value++;
        guard = 0;
    }
```

```
//6.13
monitor Demo{
    int items[MAX_ITEMS];
    int numItems = 0;
    condition full,empty;
    void producer(int v)
    {
        if( numItems == MAX_ITEMS)
        full.wait();
        items[numItems++]= v;
        empty.signal();
    }
    int consumer(){
        int reVal;
        while(numItems == 0){
            empty.wait();

        }
        reVal = items[--numItems];
        full.signal();
        return reVal;
    }
}
```







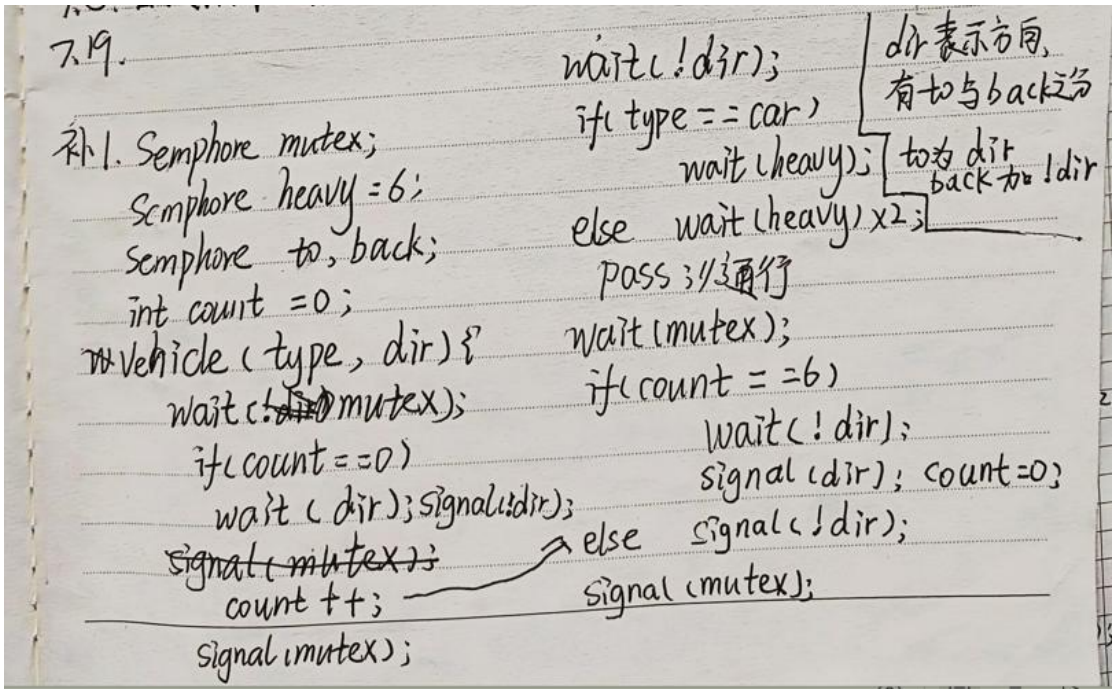
7.9//

```
package barrier1;
public class CustomBarrier {
    private final int n;
    private int count;
```

```
    private final Object lock = new Object();

    public CustomBarrier(int n) {
        this.n = n;
        this.count = 0;
    }

    public void wait1() throws InterruptedException {
        synchronized (lock) {
            count++;
            if (count == n) {
                count = 0;
                lock.notifyAll();
            } else {
                while (count != 0) {
                    lock.wait();
                }
            }
        }
    }
    public static void main(String[] args) {
        int numThreads = 5;
        CustomBarrier barrier = new CustomBarrier(numThreads);

        for (int i = 0; i < numThreads; i++) {
            new Thread(() -> {
                try {
                    System.out.println("线程 " + Thread.currentThread().getName() + " 到达屏障点
");
                    barrier.wait1();
                    System.out.println("线程 " + Thread.currentThread().getName() + " 继续执行");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }, "线程" + i).start();
        }
    }
}

package barberShop;
public class barberTest implements Runnable {
    static final private int WAIT_TIME = 3;

    static public void main(String[] args) {
```

```
        new Thread(new barberTest()).start();
    }

    public void run(){
        BarberShop newShop = new BarberShop(15);
        int customerID = 1;
        while(customerID <= 10000){
            new Thread(new Customer( newShop,customerID)).start();
            customerID++;
            SleepUtilities.nap();
        }
    }
}

package barberShop;
public class Customer implements Runnable{
    private BarberShop shop;
    private int Customer;
    private int HairCut_TIME =5;

    public Customer(BarberShop pShop, int pCustomer) {
        shop = pShop;
        Customer = pCustomer;
    }
    public void run(){
        int sleeptime = (int)(HairCut_TIME * Math.random());
        System.out.println("ENTERING SHOP: Customer [" + Customer + "] entering barber shop for
haircut.");
        int test = BarberShop.OCCUPIED;
        test = shop.getHairCut(Customer);
        if(test == BarberShop.WAITED){
            System.out.println("Barber's busy: Customer [" + Customer + "] has waited and now wants
haircut.");
        }
        else if (test == BarberShop.SLEEPING)
        System.out.println("Barber's asleep: Customer [" + Customer + "] is waking him up and getting
haircut.");
        else if (test == BarberShop.FULL){
            System.out.println("Barber Shop full: Customer [" + Customer + "] is leaving shop.");
            return;
        }
        else{
            System.out.println("HAIRCUT: Customer [" + Customer + "] is getting haircut.");
        }
```

```
        SleepUtilities.nap();
        System.out.println("LEAVING SHOP: Customer [" + Customer + "] haircut finished: leaving
shop.");
        shop.leaveBarberShop(Customer);
    }
}
package barberShop;

public class SleepUtilities {
    private static int NAP_TIME = 5;
    public static void nap(){
        nap(NAP_TIME);
    }
    public static void nap(int duration){
        int sleeptime = (int) (NAP_TIME * Math.random() );
        try { Thread.sleep(sleeptime*1000); }
        catch (InterruptedException e) {}
    }
}
```



2：
```
Semaphore mutex;
Semaphore full=4;
Semaphore empty=N-4;
int num=4;
producer(){
    x=random(2,3);
    wait(empty)
    for(int i=0;i<x;i++){
    waitp(empty);}
    wait(mutex);
    num+=x; produce xxx
    signal(mutex);
    signal(full)
    for(int i=0;i<x;i++){
     signal(full);}
    }
```

```
consumer(){
    x=min(2,num);
    for(int i=0;i<x;i++)
      wait(full);
    wait(mutex);
    num-=x; consume xxx
    for(int i=0;i<x;i++)
      signal(empty);
    signal(mutex);
```