一、Review

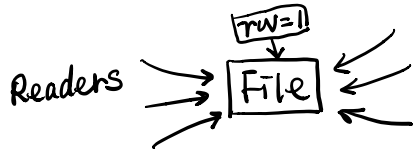
1. 工具箱 { 互斥锁 mutex lock → 临界区管理
信号量 semaphore →

2. Semaphore { 初始化值：和信号量的用途有关
P：test (waiting) > 原子操作
V：increment
init value { =1 ⇔ 互斥锁
>1 ⇒ 可用资源数量
=0 ⇒ 同步（可机动指定资源）

3. 同步问题 { 生产-消费：单缓冲和多缓冲
苹果和橘子：多种P和C，多种产品 } ⇒ 实验指导

二、同步问题案例.

1. 读者-写者问题 (Reader , Writer)



Rules:
1. R和W: 竞序
2. W和W: 竞序
3. R和R: 共享/同时.

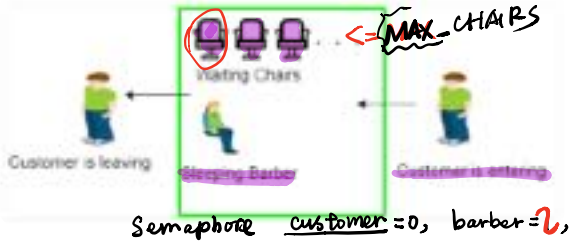
```
Semaphore  rw = 1;
int reader_count = 0; // 正在读文件的读者数量.   → 所有读者共享.
Semaphore  r_mutex = 1;

Reder_i {
    P( r_mutex);
    reader_count++;
    if (reader_count==1) {
        P(rw);
    }
    V( r_mutex);
    Read file
    P( r_mutex)
    reader_count--;
    if( reader_count ==0) {
        V(rw);
    }
    V( r_mutex);
}

Writer_i {
    P(rw)
    Write file
    V(rw)
}
```

r1 r2
1 2

2. 理发师问题.

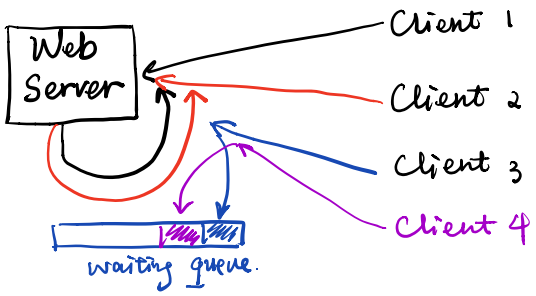



```
Semaphore  customer =0,  barber = 2,  mutex = 1;
int waiting = 0;

Barber_i{
    P(customer);
    P(mutex)
    waiting--;
    V(mutex)
    cut hair?
    V( barber);
}

Customer_i {
    P(mutex)
    if ( waiting < MAX_CHAIRS){
        waiting++;
    V(mutex)
        P(barber)
        V( customer)
        get haircut
    } else {
        V(mutex)
        Leaving...
    }
```

● 时应问计算机问题.

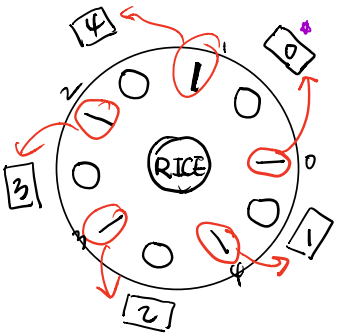


Socket
同步
异步

回调函数.
Callback.

3. 哲学家就餐问题.

Semaphore chopstick[5] = {1};




```
Philosopher_i{
  While(1){
    Thinking;//
    P(chopstick[i]);
    P(chopstick[(i+1)%5]);
    Eating;
    V(chopstick[i]);
    V(chopstick[(i+1)%5]);
  }
```

DEADLOCK. 死锁.
↓
饥饿.

The End