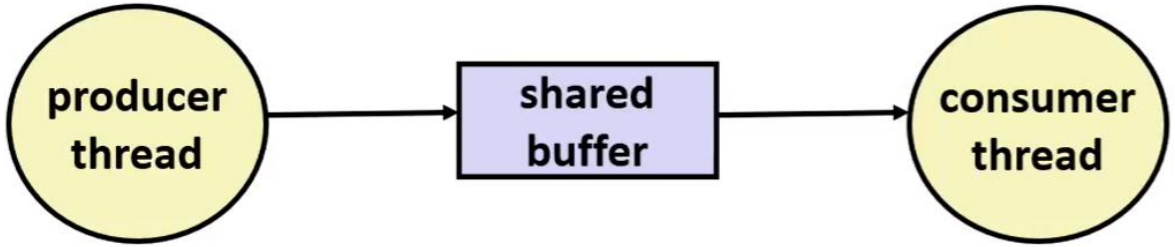
RE26

rzz

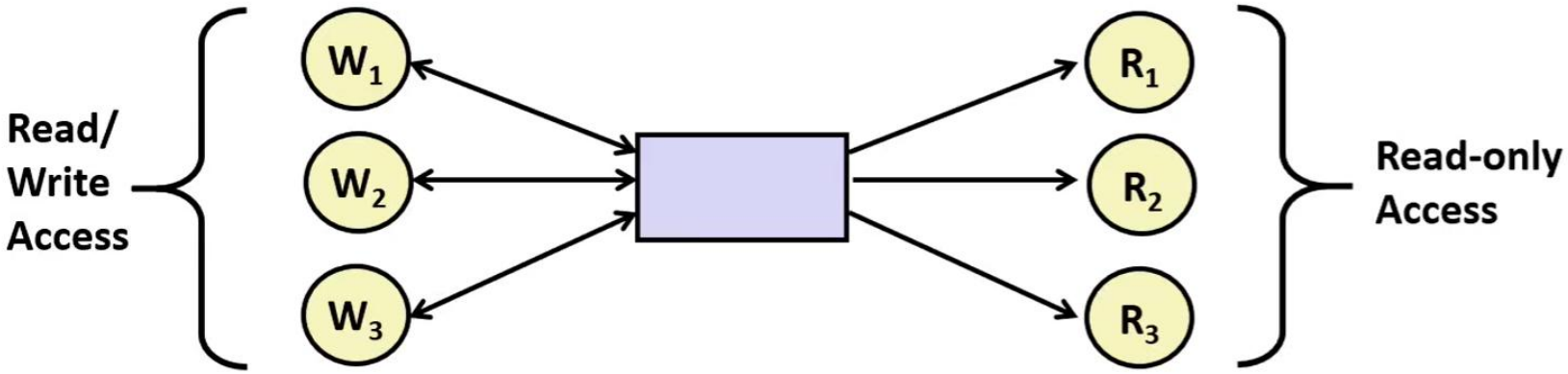
Review：信号量 semaphore

- 非负整数值的全局变量
- 两种操作：P(s) V(s)
- 两种分类：
- 二元信号量 binary semaphore
- 计数信号量 counting semaphore

- 生产者-消费者问题



Readers-Writers Problem



- Reader 可读
- Writer 可读可写
- 多个Reader可以同时进行
- Reader Writer 互斥
- Writer Writer 互斥

三种模式
- 读者优先（Favors Readers）
- 写者优先（Favors Writers）
- 先进先出（FIFO）

Readers:

```
int readcnt;    /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
  while (1) {
    P(&mutex);
    readcnt++;
    if (readcnt == 1) /* First in */
      P(&w);
    V(&mutex);

    /* Reading happens here */

    P(&mutex);
    readcnt--;
    if (readcnt == 0) /* Last out */
      V(&w);
    V(&mutex);
```

Writers:

```
void writer(void)
{
  while (1) {
    P(&w);

    /* Writing here */

    V(&w);
  }
}
```

rw1.c

竞争 Race

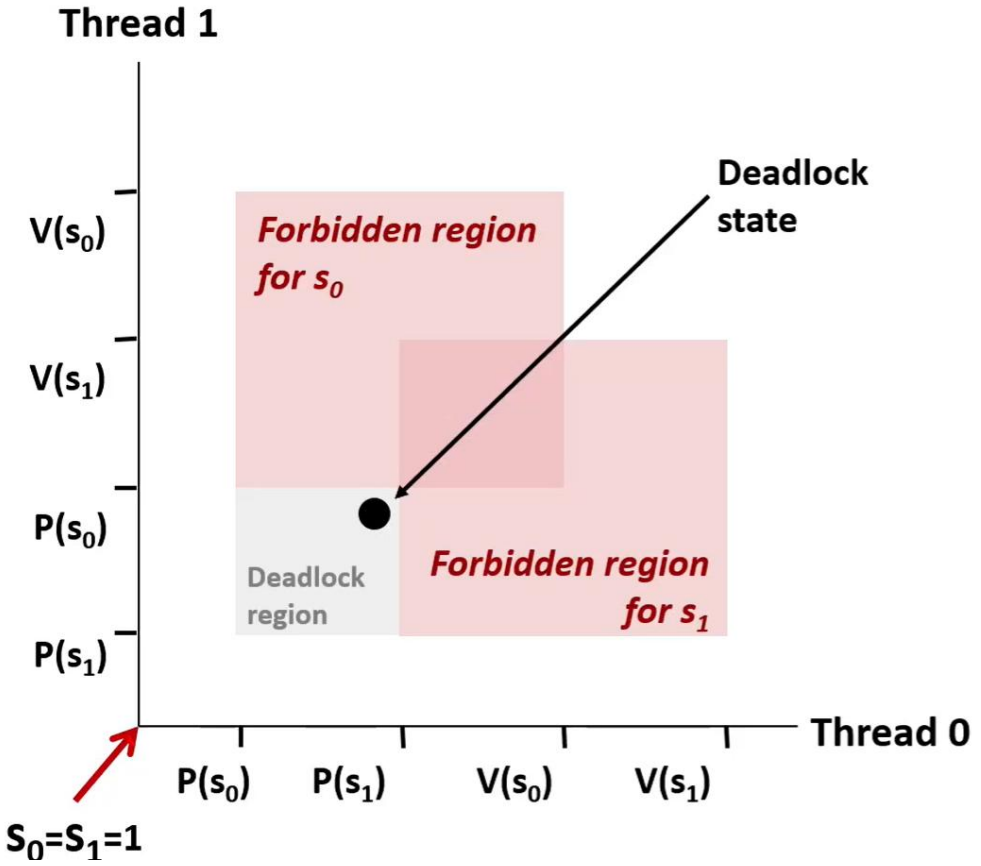
```c
/* a threaded program with a race */
int main(int argc, char** argv) {
    pthread_t tid[N];
    int i;
    for (i = 0; i < N; i++)
        Pthread_create(&tid[i], NULL, thread, &i);
    for (i = 0; i < N; i++)
       Pthread_join(tid[i], NULL);
    return 0;
}

/* thread routine */
void *thread(void *vargp) {
    int myid = *((int *)vargp);
    printf("Hello from thread %d\n", myid);
    return NULL;
}
```

解决办法：使用malloc分配独立的堆空间，不产生共享资源（记得free！）

死锁 Deadlock

```
Tid[0]:
P(s0);
P(s1);
cnt++;
V(s0);
V(s1);
```

```
Tid[1]:
P(s1);
P(s0);
cnt++;
V(s1);
V(s0);
```



互斥锁加锁规则：给定所有互斥操作的一个全序，如果每个线程都是以一种顺序获得互斥锁并以相反的顺序释放，那么这个程序就是无死锁的

线程安全

- 线程安全函数：多个线程重复调用某一函数，产生的结果均是正确的。
- 所有的标准C库函数
- 大部分Unix系统调用函数

线程安全

- 四类线程不安全函数:
- 1、未保护共享变量的函数
- 解决办法：使用P、V同步操作保护共享变量
- 缺点：降低运行效率，延长程序运行时间
- 2、当前调用结果由先前调用结果影响的函数
- 例子：伪随机数生成器rand（）
- 解决办法：重写，依靠调用者在参数中传递信息

```
/* rand_r - return pseudo-random integer on 0..32767 */

int rand_r(int *nextp)
{
    *nextp = *nextp*1103515245 + 12345;
    return (unsigned int)(*nextp/65536) % 32768;
}
```

线程安全

- 3、返回指向静态变量的指针的函数
- 解决办法：重新包装，利用互斥锁，加锁-复制技术

```
/* Convert integer to string */
char *itoa(int x)
{
    static char buf[11];
    sprintf(buf, "%d", x);
    return buf;
}
```

```
char *lc_itoa(int x, char *dest)
{
    P(&mutex);
    strcpy(dest, itoa(x));
    V(&mutex);
    return dest;
}
```

- 4、调用线程不安全函数的函数

可重入性 Reentrant

- 被多个线程调用时，不会引入任何共享数据

```
static unsigned int next = 1;

/* rand: return pseudo-random integer on 0..32767 */
int rand(void)
{
    next = next*1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}

/* srand: set seed for rand() */
void srand(unsigned int seed)
{
    next = seed;
}
```