操作系统

Operating system

孔维强

大连理工大学

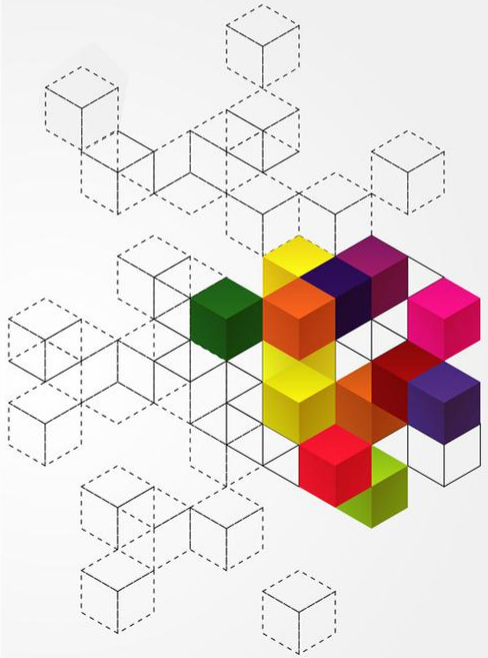
内容纲要

6.4 互斥的硬件实现

一、 引入互斥硬件解法的意义

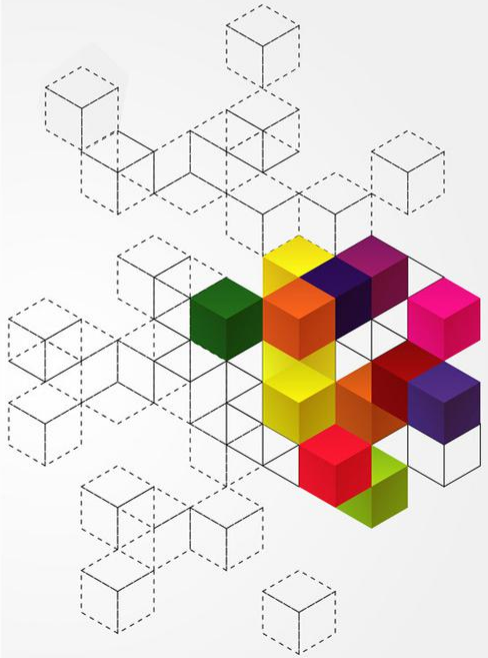
二、 互斥硬件解法的思路

三、 基于加锁操作的互斥实现



一、引入互斥硬件解法的意义

- **互斥，用软件方法实现代价非常大**
  - 一般性的多进程解法（面包店算法），其进入区保护代码的实现成本很高，无法实际应用
- 互斥实现的最大难题
  - 判断进入临界区条件，同时设置本进程已经进入临界区的状态，这两个操作之间可能存在时间窗



二、互斥硬件解法的思路

解决互斥问题的最直观实现

lock = 0

获取锁-CS-释放锁

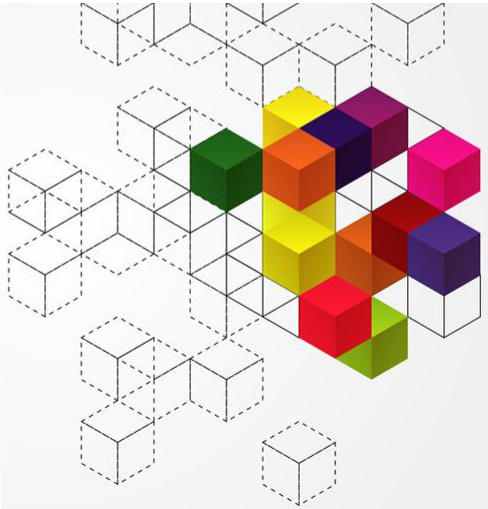
进程A

```
While(lock != 0)
        NULL;
lock = 1;
Critical_region();
lock = 0;
```

进程B

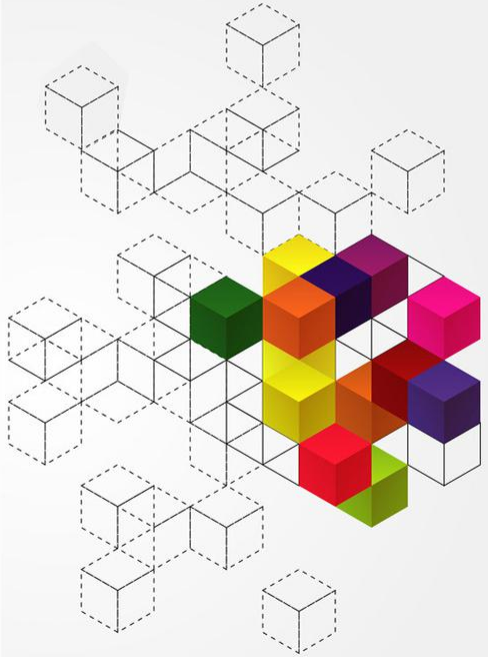
```
While(lock != 0)
        NULL;
lock = 1;
Critical_region();
lock = 0;
```

```
do {
        acquire lock
                critical section
        release lock
                remainder section
} while (TRUE);
```



二、互斥硬件解法的思路

- 互斥问题的直观加锁解决方案的问题
  - 在检查临界区是否被上锁（while循环），以及获准进入后上锁(lock=1)之间存在一个时间窗



二、互斥硬件解法的思路

- 互斥问题的直观加锁解决方案的问题（问题调度）

lock = 0

进程A

While(lock != 0);

进程B

While(lock != 0);

进程A

lock = 1;

Critical_Region();

进程B

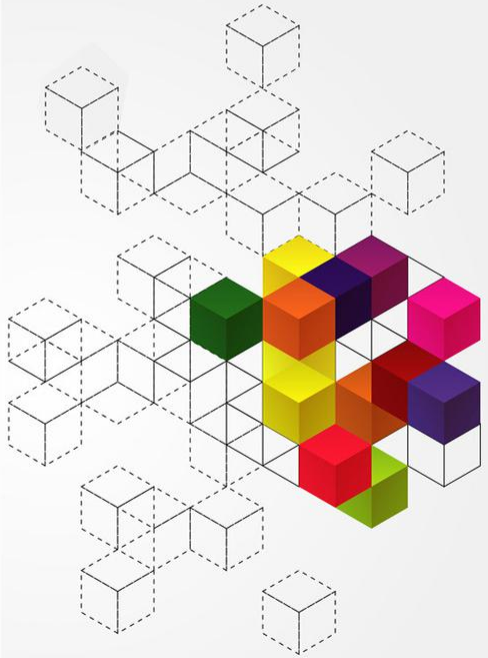
lock = 1;

Critical_Region();

发生进程调度，互斥的进程开始运行

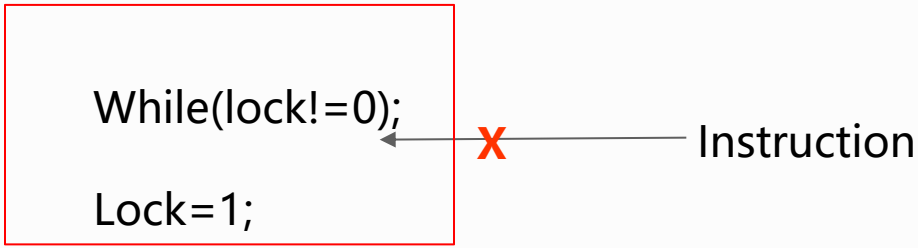
发生进程调度，A并不知道B也进入临界区

发生进程调度，B并不知道A也进入临界区

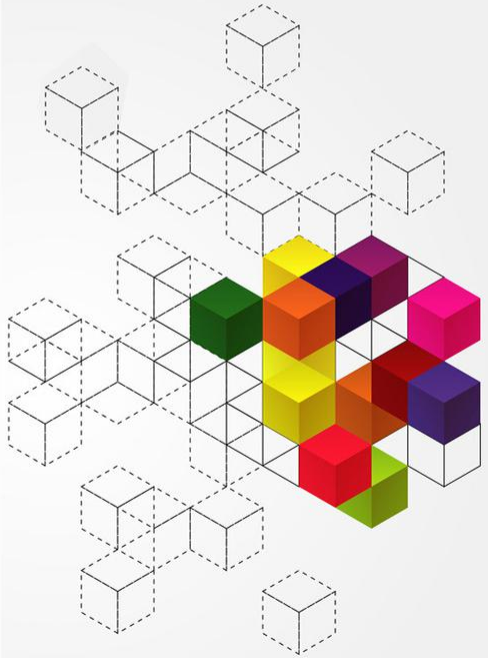


二、互斥硬件解法的思路

- 互斥问题的直观加锁解决方案的问题



  - 保证while循环，lock置1的操作一起执行，不需被中断。
  - 称为原子操作（Atomic Operation）

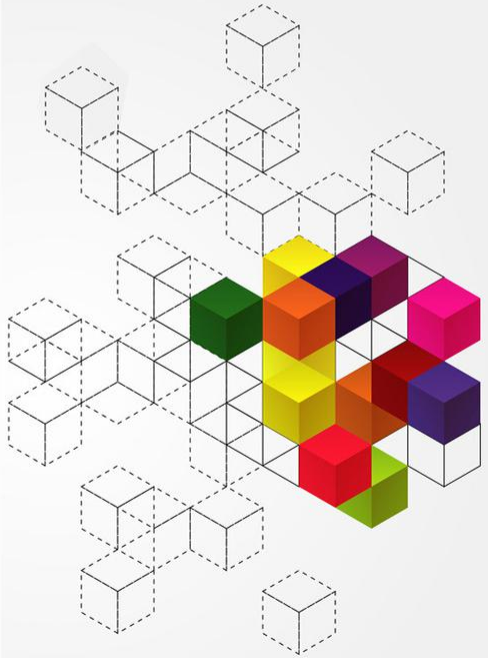


二、互斥硬件解法的思路

- 最终实现思路归结为：提供TestAndSet指令
  - 使用TestAndSet来处理上述临界区问题
  - TestAndSet(lock)指令语义：
    - lock=0, 则将lock置1，返回0
    - lock=1, 则直接返回1

Definition:

```
boolean test_and_set (boolean *target)
  {
        boolean rv = *target;
        *target = TRUE;
        return rv:
  }
```



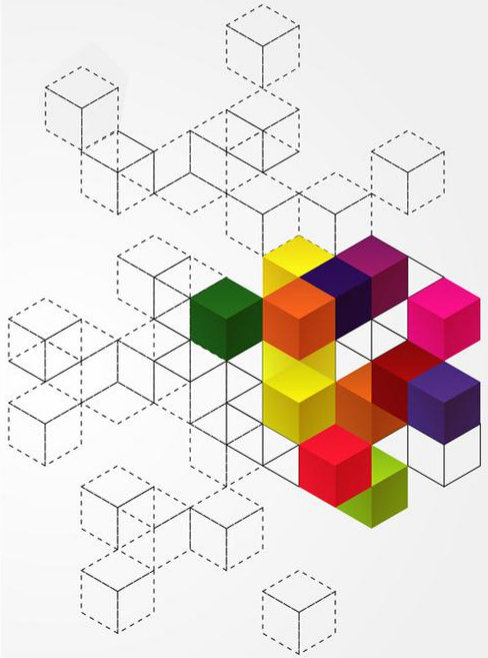
三、基于加锁操作的互斥实现

临界区通过基于TestAndSet原子操作实现的锁对象保护，方式如下

```
do{
    while(TestAndSet(lock));
    临界区
    lock = false;
    剩余区;
}while(1);
```

满足临界区条件(1)和(2):
-互斥
-空闲让进（前进）

不满足临界区条件（3）：非有限等待

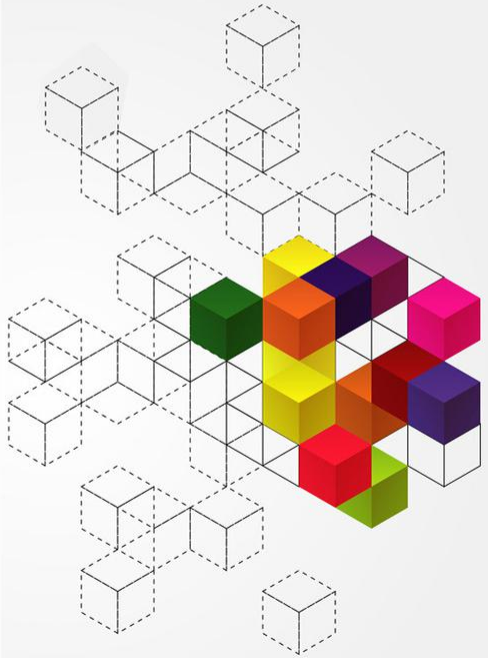


三、基于加锁操作的互斥实现

- 临界区通过基于Swap原子操作实现的锁对象保护，方式如下

Definition:

```
void Swap (boolean *a, boolean *b)
        {
                boolean temp = *a;
                *a = *b;
                *b = temp:
        }
```

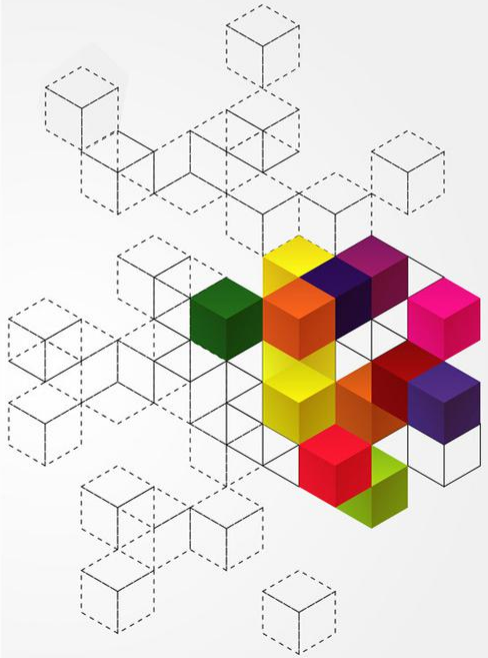


三、基于加锁操作的互斥实现

临界区通过基于Swap原子操作实现的锁对象保护，方式如下

- 共享变量 lock 初值为 FALSE；每个进程有局部布尔变量 key.
- Solution:

```
do {
    key = TRUE;
    while ( key == TRUE)  Swap (&lock, &key );
    //    critical section
    lock = FALSE;
    //    remainder section
} while ( TRUE);
```



三、基于加锁操作的互斥实现

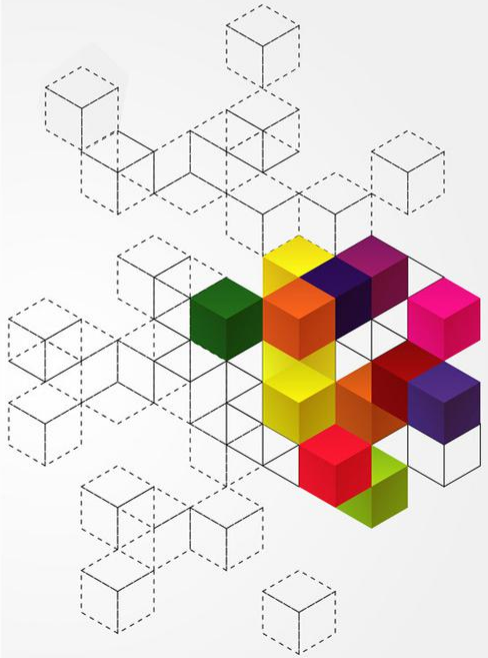
```
do{
    waiting[i] = true
    key = true;
    while(waiting[i]&&key)
        key=TestAndSet(lock);
    waiting[i]=false;

    临界区

    j=(i+1)%n;
    while(j!=i && !waiting[j])
        j = (j+1)%n;
    if(j==i)
        lock = false;
    else
        waiting[j] = false;
     剩余区
}while(1);
```

表示进程Pi处于等待获取锁的状态

如果进程Pi抢到了锁，记录key=false

进程Pi处于等待获取锁的状态

共享变量及初值：
wait[i]=false;
lock=false;



本讲小结

- 引入互斥硬件解法的意义
- 互斥硬件解法的思路
- 基于加锁操作的互斥实现