操作系统

Operating system

孔维强

大连理工大学

内容纲要

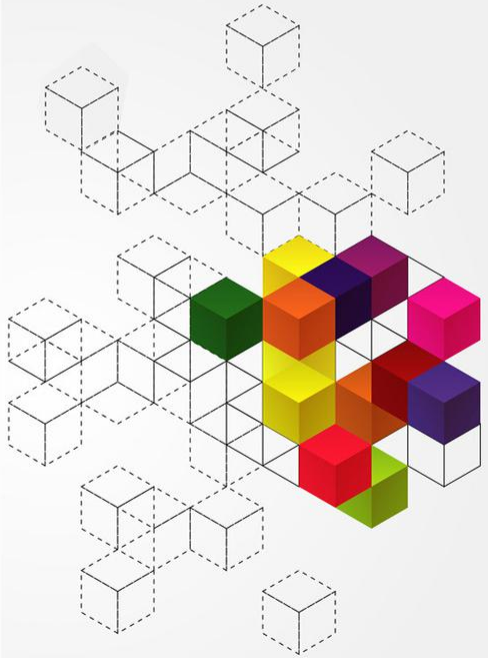
6.5 信号量概念

一、 引入信号量的动机

二、 信号量概念

三、 信号量实现

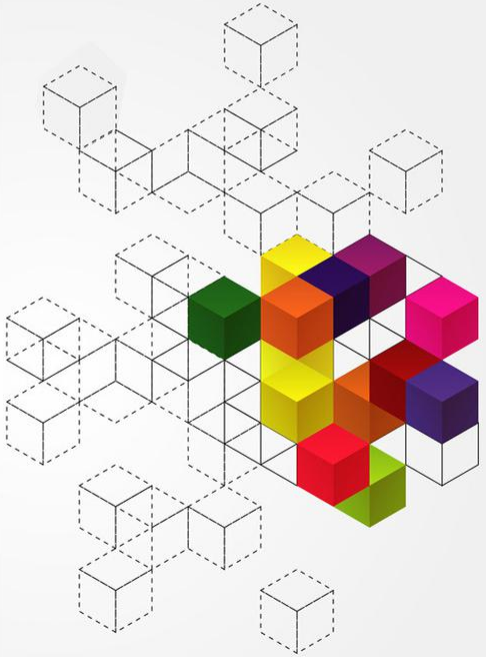
四、 信号量编程接口示例



一、引入信号量的动机

为什么需要信号量：

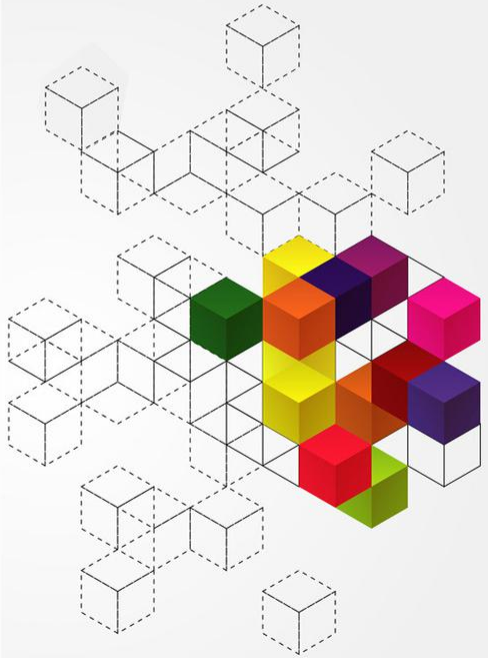
- 面包店算法，是实现互斥的一般性软件解法
- 当多个进程竞争使用的某类资源具有多个资源实例时，互斥性如何保证
- 例如：卡拉OK房间内的麦克风（2个），唱歌时竞争使用资源的状态就会有：
- 1个人独唱
- 2个人合唱，其余人休息
- 2个人合唱，1个或多个人等待唱歌



一、引入信号量的动机

分析一下一般情况下的多资源实例竞争问题

- 初始状态：剩余资源数available = n
- 中间状态：陆续有k(<n)个进程，每个进程取走一个资源，那么available=n-k
- 临界状态：某个时刻，最后一个资源可能被取走，那么available=0
- 有q个进程在等待资源的状态

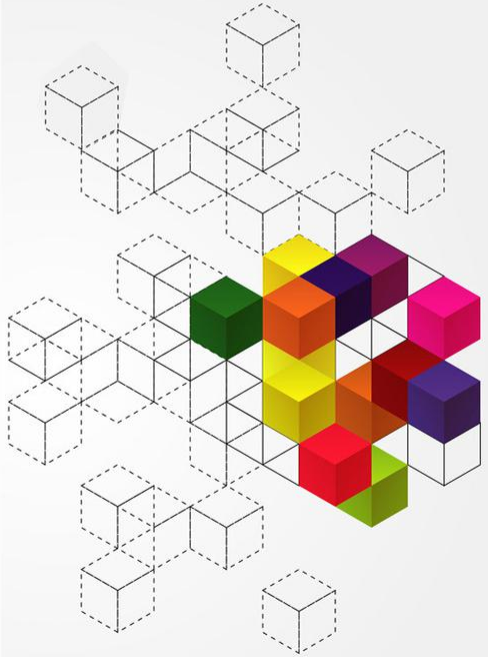


二、信号量概念

信号量（Semaphore）

- 一个信号量S是一个整型量，除对其初始化外，它只能由两个原子操作P和V来访问

- 1965年，荷兰科学家Dijkstra提出
- P和V的名称来源于荷兰文proberen(测试)和verhogen(增量)
- 亦有将P/V操作分别称作wait(), signal()



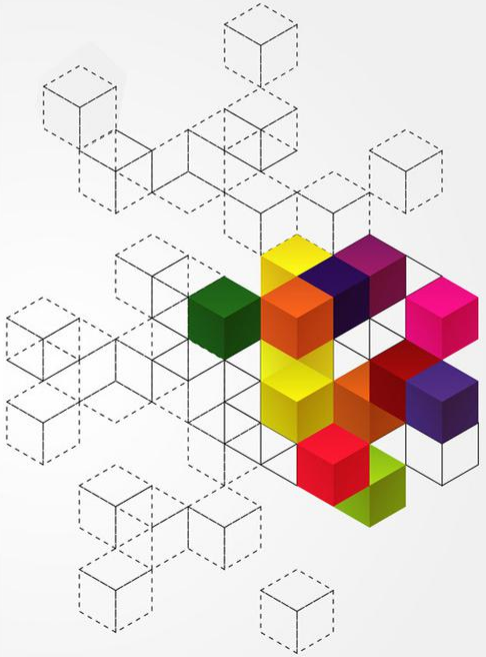
二、信号量概念

信号量（Semaphore）

- 信号量的P，V操作的简单伪代码
- 表达了P,V操作的基本语义，但是并不是实际的实现方式

```
Wait(){
    while(S<=0) ; //忙等
    S--;
}
```

```
Signal(){
    S++;
}
```

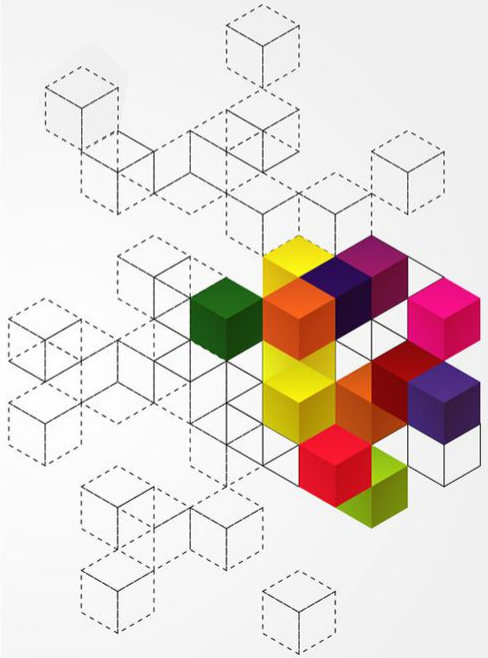


二、信号量概念

信号量（Semaphore）

- 使用信号量解决临界区问题，mutex初值为1

```
Do{
  waiting(mutex);
    // critical section
  signal(mutex);
   //   remainder section
  }while(TRUE);
```



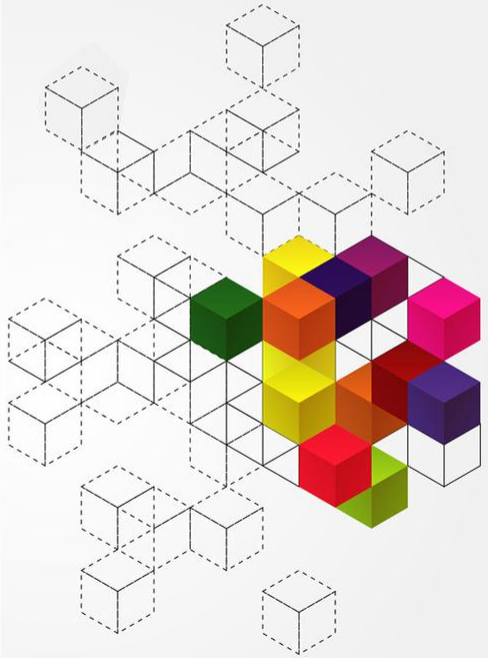
二、信号量概念

信号量（Semaphore）

```
Wait(){
    while(S<=0) ;
    S--;
}
```

```
Signal(){
    S++;
}
```

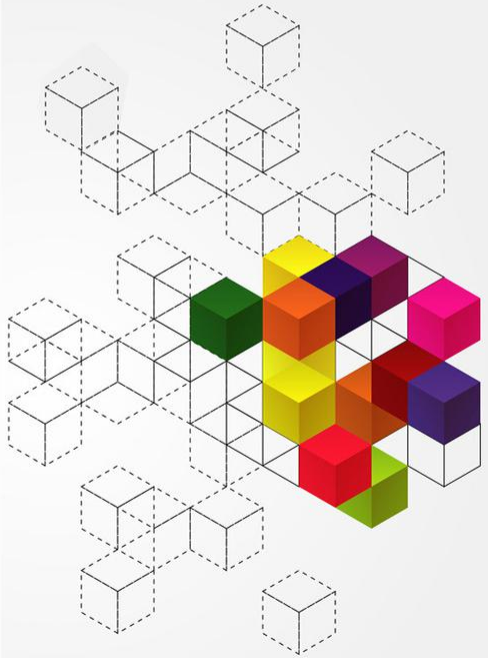
- 上述简单伪代码中的不合实际的问题：
- (1)不能忙等
- (2)没有办法记录处于等待状态的进程数量



三、信号量实现

避免进程忙等，wait和signal的定义需要进行修改

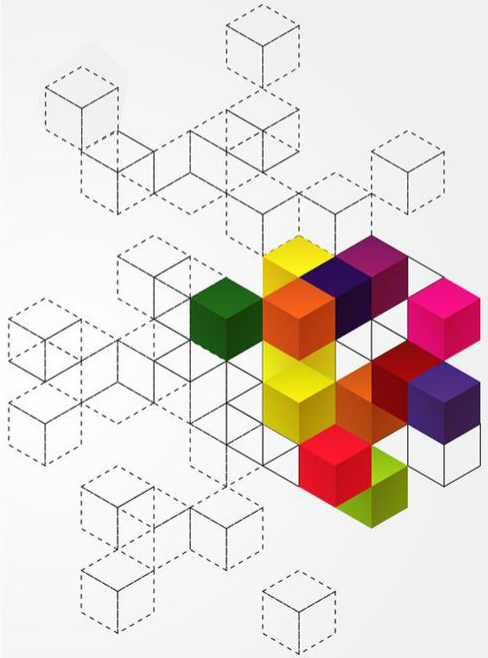
- Wait:
- 当一个进程执行wait操作但发现信号量S<=0时，它必须等待，这里的等待不是忙等，而是阻塞自己
- 阻塞操作将一个进程放入与信号量相关的等待队列中，且该进程的状态被切换成等待状态，接着控制被转到CPU调度程序，以选择另一个进程来执行



三、信号量实现

避免进程忙等，wait和signal的定义需要进行修改

- Signal:
- 一个进程阻塞且等待信号量S，可以在其他进程执行signal操作之后被重新执行



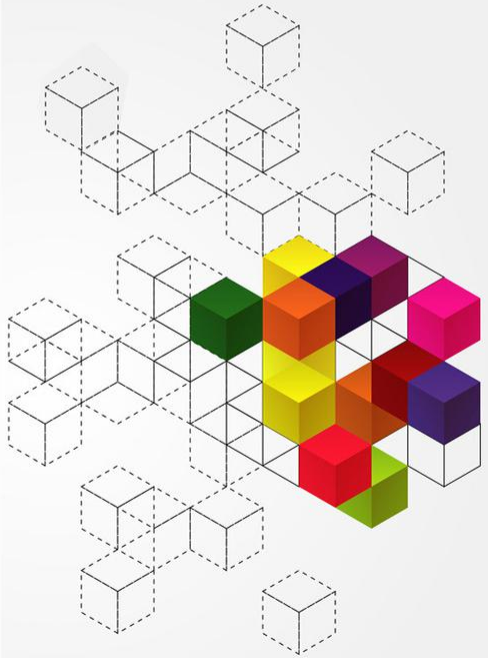
三、信号量实现

信号量数据结构

- 为了定义基于阻塞（block）/唤醒（wakeup）的信号量，可以将信号量定义为如下结构

```
typedef  struct {
    int value;
    struct process *L;
}semaphore;
```

在该信号量上阻塞的进程队列

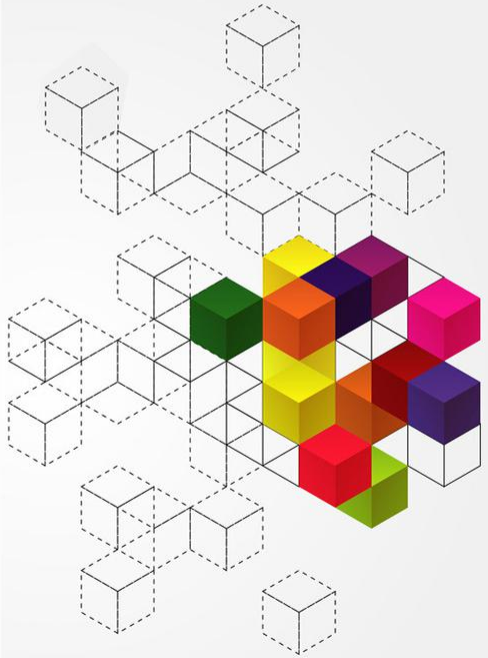


三、信号量实现

- 合理的P操作实现

```
void  wait(semaphore S){
    S.value--;
    if(S.value<0){
        add this process to S.L;
        block();
    }
}
```

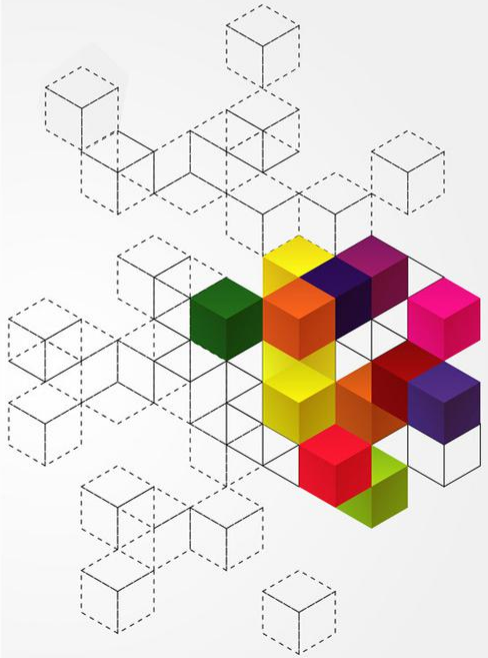
S的值可为负，其值表示多少进程在等待



三、信号量实现

- 合理的V操作实现

```
void  signal(semaphore S){
    S.value++;
    if(S.value<=0){
        remove a process P from S.L;
        wakeup(P);
    }
}
```



三、信号量实现

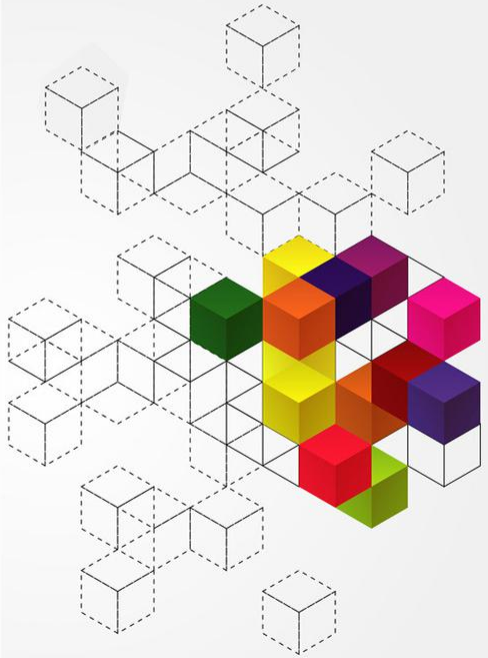
死锁和饥饿问题

✓ 死锁：两个或多个进程无限等待一个事件的发生，而该事件仅能由处于等待的进程产生

P_0	P_1
① wait(S);	② wait(Q);
③ wait(Q);	④ wait(S);
...	...
signal(S);	signal(Q);
signal(Q);	signal(S);

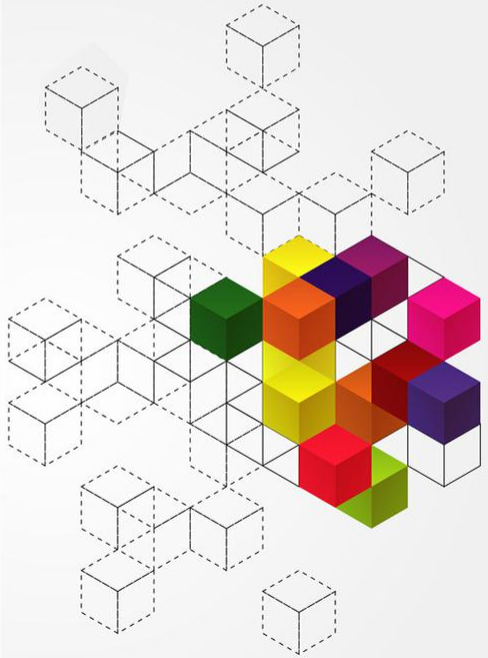
✓ 饥饿：无限阻塞

死锁意味着饥饿，但饥饿不一定意味着死锁



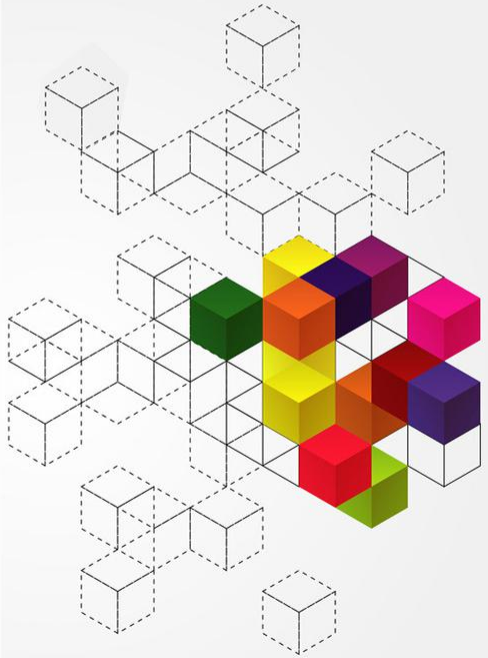
四、信号量编程接口示例

```
// C program to demonstrate working of Semaphores
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;
```



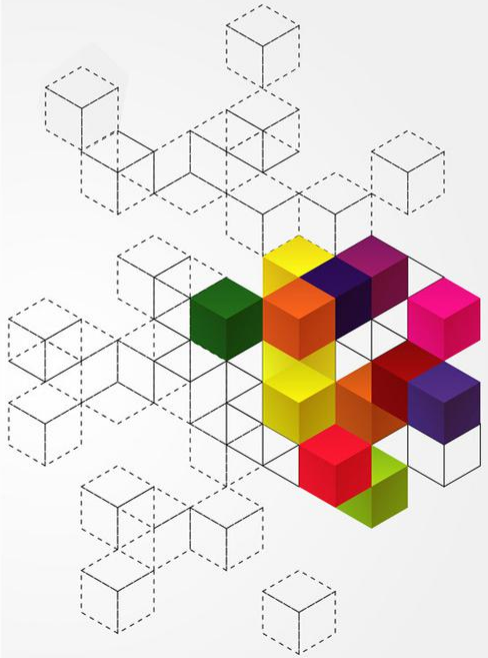
四、信号量编程接口示例

```
void* thread(void* arg)
{
    //wait
    sem_wait(&mutex);
    printf("\nEntered..\n");

    //critical section
    sleep(4);

    //signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}
```



四、信号量编程接口示例

```
int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```



本讲小结

- 信号量的引入
- 信号量概念
- 信号量实现
- 信号量编程接口示例