操作系统

Operating system

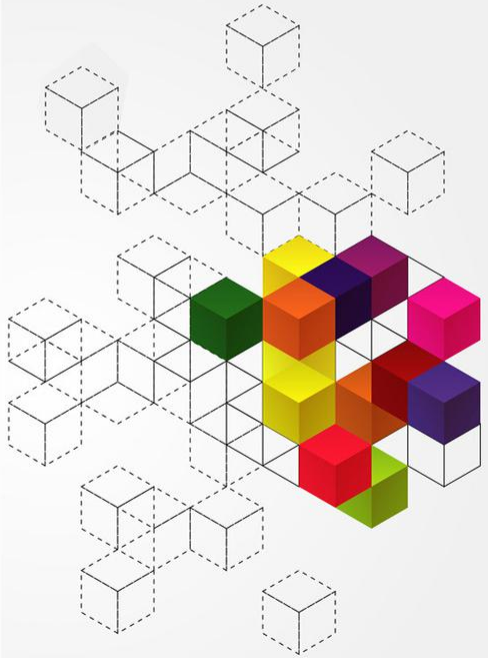
孔维强

大连理工大学

一、 进程同步概念

二、 进程直接协作

三、 进程间接协作



一、进程同步概念

- Process Synchronization

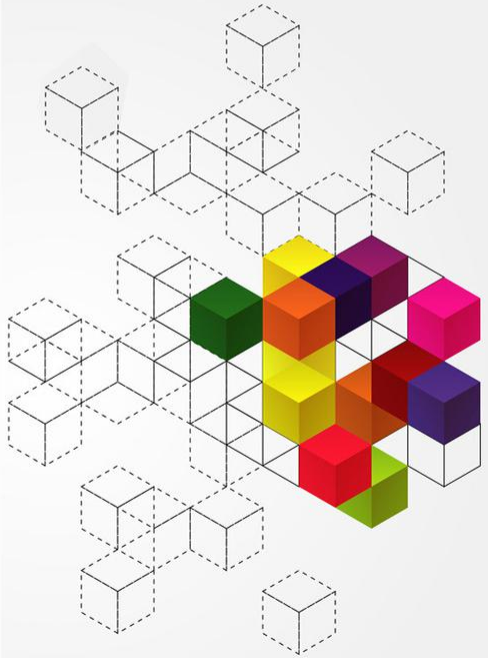
现代操作系统的2大亮点：

1.进程在独立的地址空间内进行计算任务

2.不同进程异步执行

实现多任务；高效执行

为什么又需要Synchronization呢?



一、进程同步概念

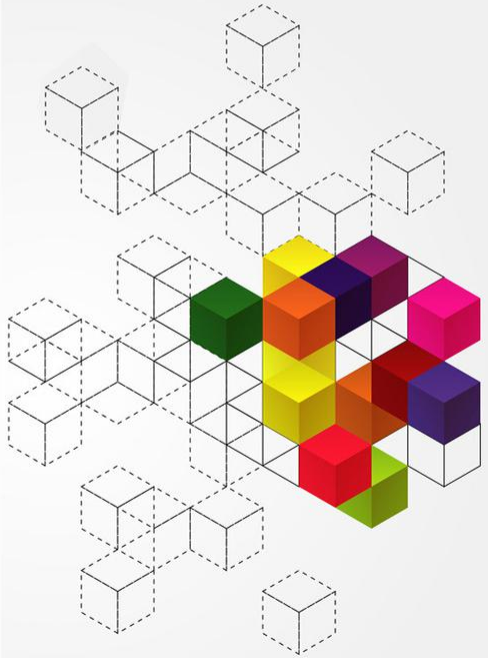
- **Why Process Synchronization?**

现代操作系统上的应用日益复杂

多任务是常态

- 模块化划分后，用多进程、多线程实现，充分利用多核算力
- 任务之间需要传递信息进行协同
  任务更频繁涉及I/O：文件I/O，网络I/O等
- 不同任务可能同时访问相同设备

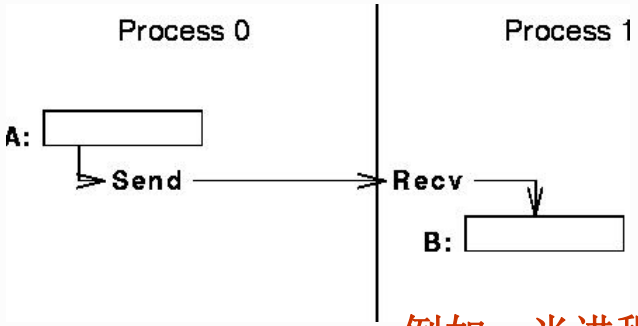
现代应用程序：以异步为基础，同步为辅助



一、进程同步概念

- 进程协作的两种类型:直接协作和间接协作

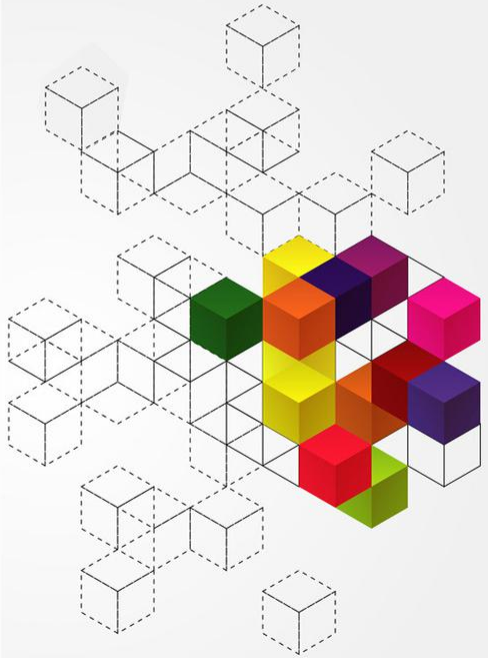
直接协作（Direct Cooperation）



例如：当进程运行到某一点时要求另一伙伴进程为它提供消息

在未获得消息之前，该进程处于等待状态

获得消息后被唤醒进入就绪态



二、进程直接协作

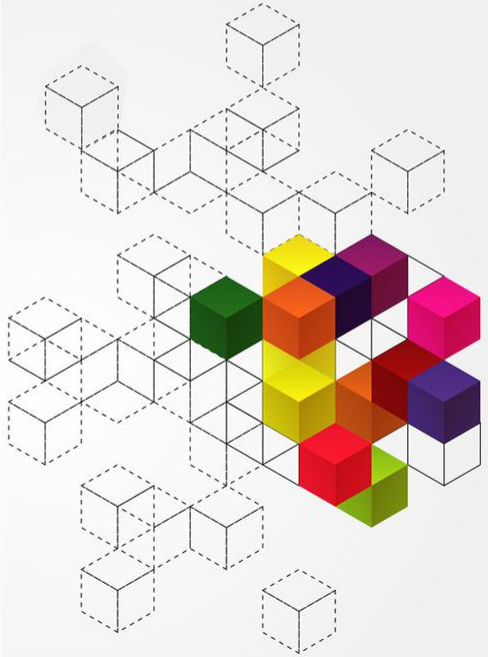
- Direct Cooperation： 示例
  - 司机与售票员之间的协作

司机 P_1

```
While(true)
{
    启动车辆;
    正常运行;
    到站停车;
}
```

售票员 P_2

```
While(true)
{
    关门;
    售票;
    开门;
}
```



一、进程同步概念

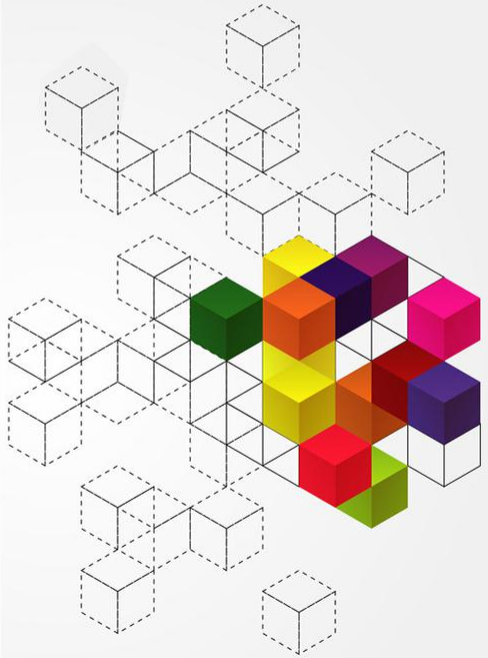
- 进程协作的两种类型:直接协作和间接协作

直接协作（Direct Cooperation）

间接协作（Indirect Cooperation）

进程之间间接作用

多个进程竞争使用共享数据（资源）
间接作用不会强制合作进程之间遵循特定的先后顺序
间接合作进程必须保证对共享数据的一致性访问



三、进程间接协作

- 进程之间间接作用示例

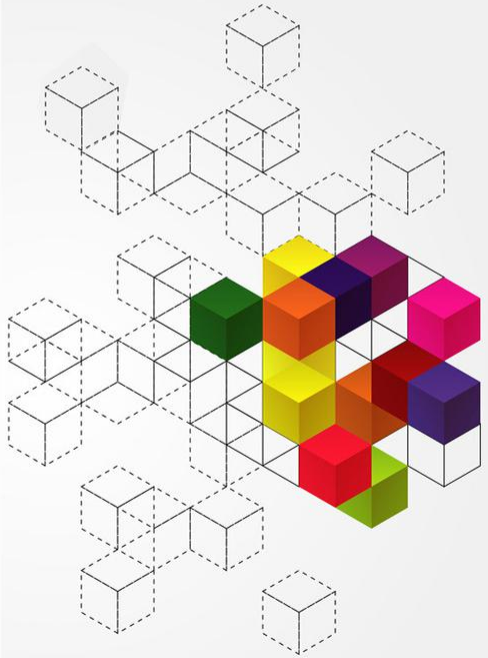
P_1:

```
if(x>=100){
    x-=100;
}
```

P_2:

```
if(x>=100){
    x-=100;
}
```

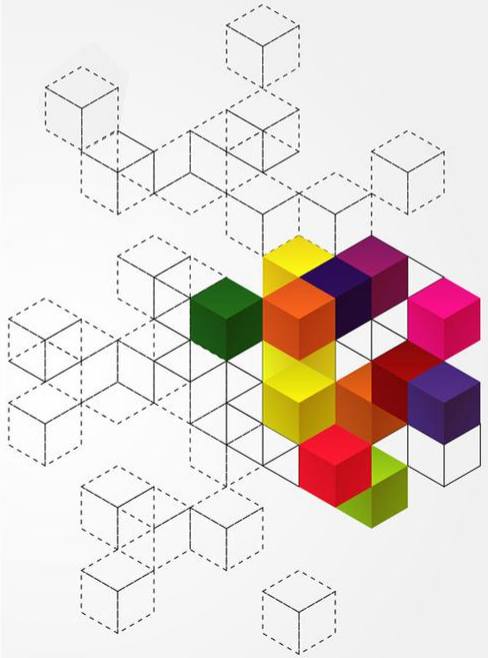
x为共享变量，初值=100



三、进程间接协作

- 多进程可并发执行
  - 进程的执行可能在任意时间点被中断，导致部分执行
- 对于共享数据的并发访问可能导致数据的不一致性
- 维护数据的一致性需要一定的机制

- 问题示例：
  - 考虑生产者-消费者问题，使用count变量记录被使用的缓冲区个数，其初值为0，生产商品后值加1，消耗商品后值减1.

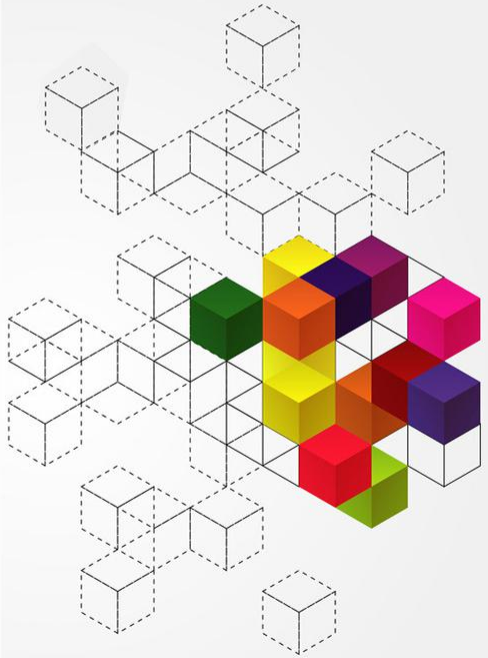


三、进程间接协作

- Producer（生产者）

```
while (true) {
        /* produce an item in next produced */

        while (counter == BUFFER_SIZE) ;
                /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

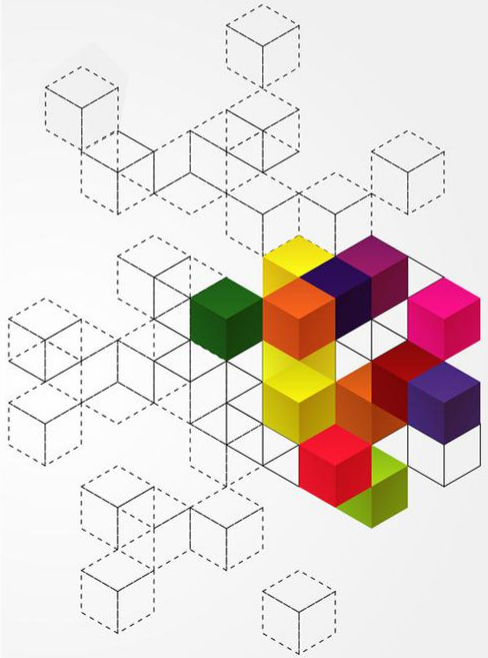


三、进程间接协作

- 消费者（Consumer）

```
while (true) {
        while (counter == 0)
              ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
           counter--;
        /* consume the item in next consumed */
}
```

Counter++ 与 Counter--并发执行会产生什么结果？



三、进程间接协作

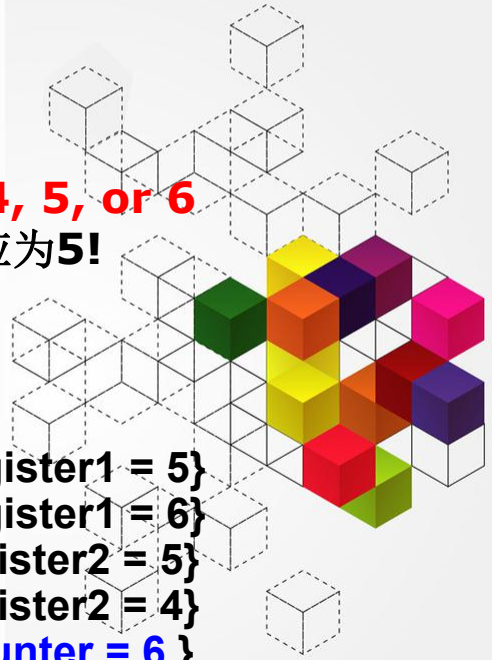
■**counter++** 可被实现为机器语言（register1 & 2为CPU寄存器）

```
register1 = counter
register1 = register1 + 1
counter = register1
```

■**counter--**可被实现为机器语言

```
register2 = counter
register2 = register2 - 1
counter = register2
```

Counter可能为4, 5, or 6
但其正确值应为5!



■假设counter的初值为5，并发执行的可能顺序:

S0: producer execute `register1 = counter` **{register1 = 5}**
S1: producer execute `register1 = register1 + 1` **{register1 = 6}**
S2: consumer execute `register2 = counter` **{register2 = 5}**
S3: consumer execute `register2 = register2 - 1` **{register2 = 4}**
S4: producer execute `counter = register1` **{counter = 6 }**
S5: consumer execute `counter = register2` **{counter = 4}**

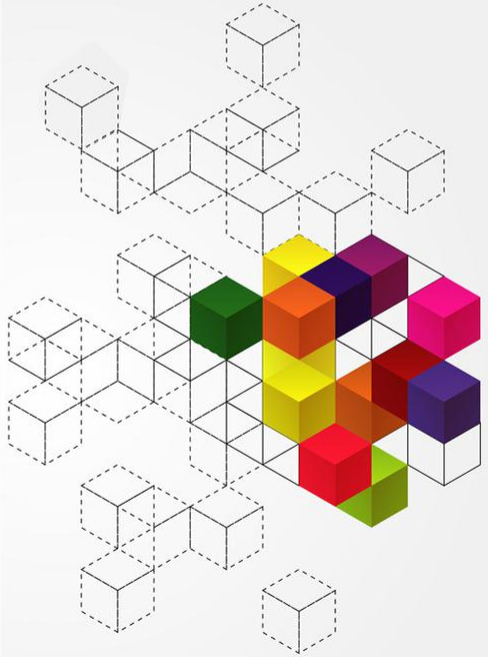
三、进程间接协作

- 竞争条件（race condition）

  多个进程并发访问并操作共享数据，结果值依赖特定的访问顺序

- 为防护竞争条件的发生，每次只允许一个进程访问共享数据

- 进程同步（process synchronization）处理此类问题



本讲小结

- 进程同步概念
- 进程直接协作
- 进程间接协作