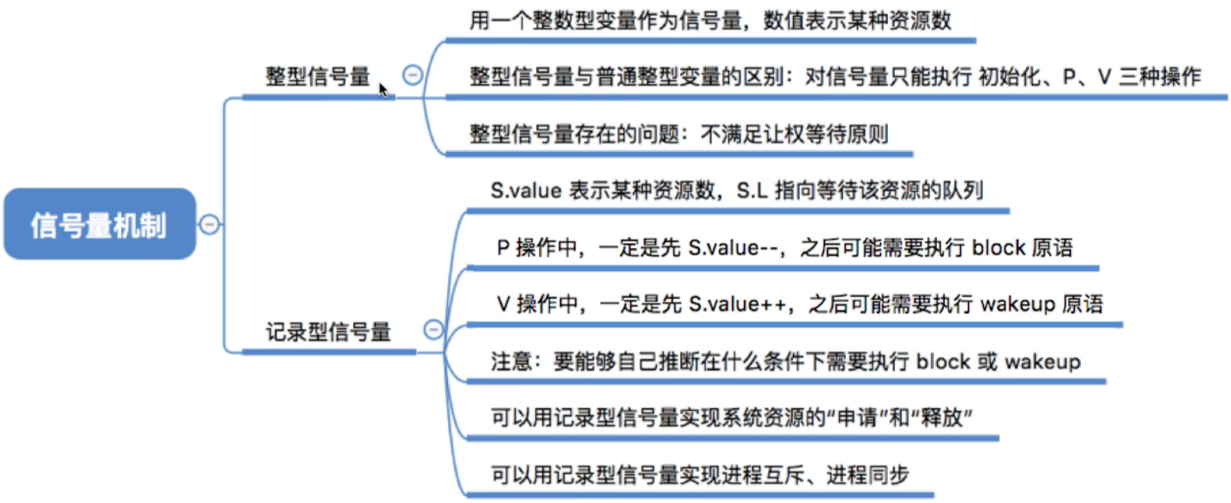
信号量



之前学习的这些进程互斥的解决方案分别存在哪些问题？

进程互斥的四种软件实现方式（单标志法、双标志先检查、双标志后检查、Peterson算法）

进程互斥的三种硬件实现方式（中断屏蔽方法、TS/TSL指令、Swap/XCHG指令）

1. 在双标志先检查法中，进入区的“检查”、“上锁”操作无法一气呵成，从而导致了两个进程有可能同时进入临界区的问题
2. 所有的解决方案都无法实现“让权等待”

1965年，荷兰学者Dijkstra提出了一种卓有成效的实现进程互斥、同步的方法——信号量机制

用户进程可以通过使用操作系统提供的一对原语来对信号量进行操作，从而很方便的实现了进程互斥、进程同步。

信号量其实就是一个变量（可以是一个整数，也可以是更复杂的记录型变量），可以用一个信号量来表示系统中某种资源的数量，比如：系统中有一台打印机，就可以设置一个初值为1的信号量。

原语是一种特殊的程序段，其执行只能一气呵成，不可被中断。原语是由关中断/开中断指令实现的。软件解决方案的主要问题是由“进入区的各种操作无法一气呵成”，因此如果能把进入区、退出区的操作都用“原语”实现，使这些操作能“一气呵成”就能避免问题。

一对原语：wait(S)原语和signal(S)原语，可以把原语理解为我们自己写的函数，函数名分别为wait和signal，括号里的信号量S其实就是函数调用时传入的一个参数。

wait、signal原语常简称为P、V操作（来自荷兰语proberen和verhogen）。因此，做题的时候常把wait(S)、signal(S)两个操作分别写为P(S)、V(S)

整型信号量

用一个整数型的变量作为信号量，用来表示系统中某种资源的数量。

与普通整数变量的区别：对信号量的操作只有三种，即初始化、P操作、V操作

"检查"和“上锁”一气呵成，避免了并发、异步导致的问题

```
int S=1;//初始化整型信号量S，表示当前系统中可用的打印机资源数
void wait(int S){//wait原语，相当于"进入区"
    while(S<=0);//如果资源数不够，就一直循环等待
    S=S-1;//如果资源数够，则占用一个资源
}
void signal(int S){//signal原语，相当于“退出区”
    S=S+1;//使用完资源后，在退出区释放资源
}
```

```
//进程P0
//...
wait(S);//进入区，申请资源
//使用打印机资源...临界区，访问资源
signal(S);//退出区，释放资源
//...

//进程P1
//...
wait(S);
//使用打印机资源...
signal(S);
//...

//进程Pn
//...
wait(S);
//使用打印机资源...
signal(S);
//...
```

存在的问题：不满足“让权等待”原则，会发生“忙等”

记录型信号量

整型信号量的缺陷是“忙等”问题，因此人们又提出了“记录型信号量”，即用记录型数据结构表示的信号量。

```
//记录型信号量的定义
typedef struct{
    int value;//剩余资源数
    struct process *L;//等待队列
}semaphore;
```

```
//某进程需要使用资源时，通过wait原语申请
void wait(semaphore S){
    S.value--;
    if(S.value<0){
        //如果剩余资源数不够，使用block原语使进程从运行态进入阻塞态，并把挂到信号量S的等待队列（即阻塞队列）中
        block(S.L);
    }
```

```
}
//进程使用完资源后，通过signal原语释放
void signal(semaphore S){
    s.value++;
    if(S.value<=0){
        //释放资源后，若还有别的进程在等待这种资源，则使用wakeup原语唤醒等待队列中的一个进程，该进程从阻塞态变为就绪态
        wakeup(S.L);
    }
}
```

$$
\begin{aligned}
S.value+1 \leq 0, &\text{说明有进程在等待该资源} \\
S.value = 0, &\text{资源恰好分配完} \\
S.value = -1, &\text{有1个进程在等待} \\
S.value = -2, &\text{有2个进程在等待} \\
S.value > 0, &\text{说明已没有进程在等待该资源}
\end{aligned}
$$

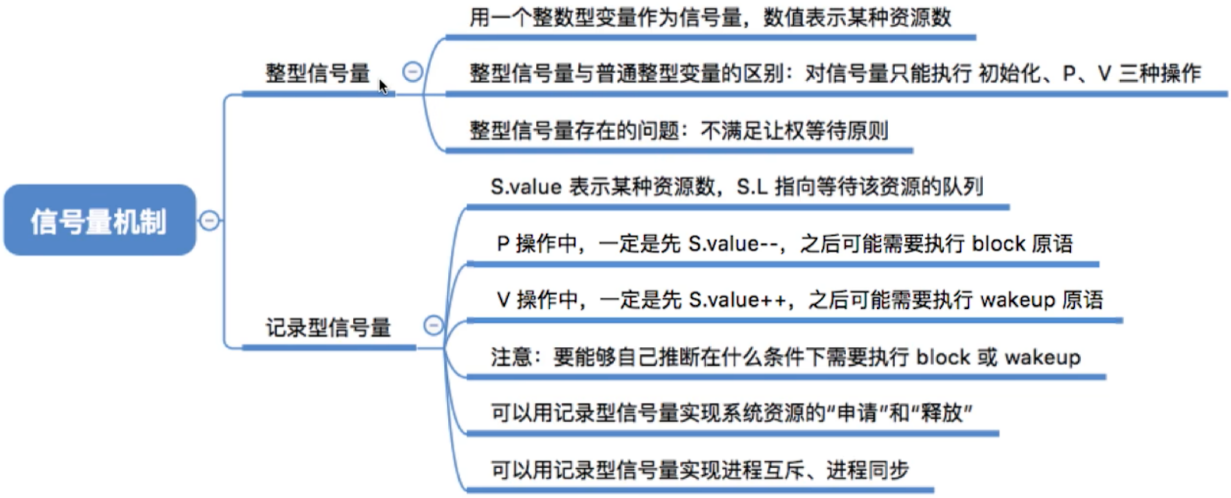
wait(S), Signal(S)也可以记为P(S), V(S)，这对原语可用于实现系统资源的“申请”和“释放”。

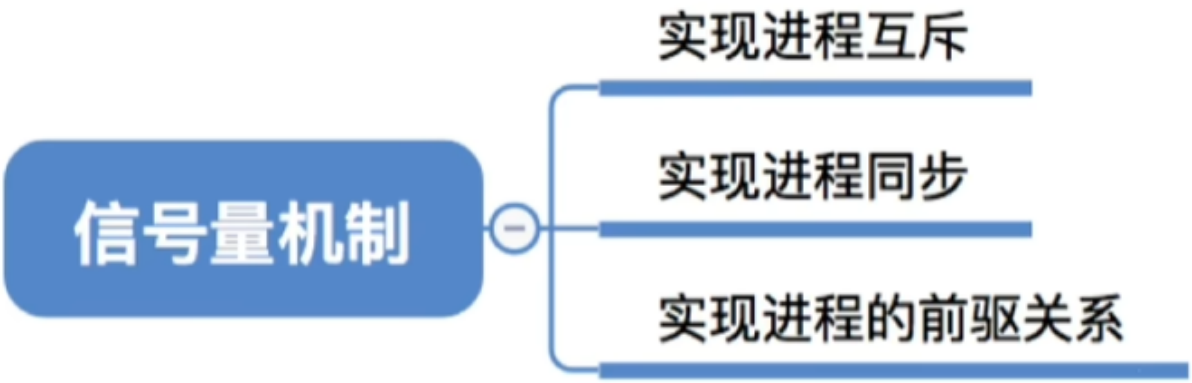
S.value的初值表示系统中某种资源的数目。

对信号量S的一次P操作意味着进程请求一个单位的该类资源，因此需要执行S.value--，表示资源数减1，当S.value<0时表示该类资源已分配完毕，因此进程应调用block原语进行自我阻塞（当前运行的进程从运行态->阻塞态），主动放弃处理机，并插入该类资源的等待队列S.L中。可见，该机制遵循了“让权等待”原则，不会出现“忙等”现象。

对信号量S的一次V操作意味着进程释放一个单位的该类资源，因此需要执行S.value++，表示资源数加1，若加1后仍是S.value<=0，表示依然有进程在等待该类资源，因此应调用wakeup原语唤醒等待队列的第一个进程（被唤醒进程从阻塞态->就绪态）。

- 互斥问题，信号量初值为1
- 同步问题，信号量初值为0
- 前驱关系问题，本质上就是多级同步问题
- 除了互斥、同步问题外，还会考察有多个资源的问题，有多少资源就把信号量初值设为多少。申请资源时进行P操作，释放资源时进行V操作即可





一个信号量对应一种资源

$$\text{信号量的值} = \text{这种资源的剩余量}(\text{信号量的值如果小于}0,\text{说明此时有进程在等待这种资源})$$

P(S)——申请一个资源S，如果资源不够就阻塞等待

V(S)——释放一个资源S，如果有进程在等待该资源，则唤醒一个进程

信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量mutex，初值为1，信号量mutex表示"进入临界区的名额"
3. 在进入区P(mutex)——申请资源
4. 在退出区V(mutex)——释放资源

注意：对不同的临界资源需要设置不同的互斥信号量。

P、V操作必须成对出现。缺少P(mutex)就不能保证临界资源的互斥访问。缺少V(mutex)会导致资源永不被释放，等待进程永不被唤醒。

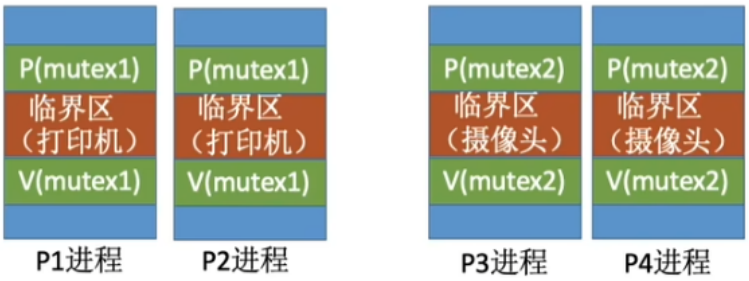
```
//记录型信号量的定义
typedef struct{
    int value;//剩余资源数
    struct process *L;//等待队列
}semaphore;
```

```
//信号量机制实现互斥
semaphore mutex = 1;//初始化信号量，要会自己定义记录型信号量，但如果题目中没特别说明，可以把信号量的声明简写成这种形式

P1(){
    //...
    P(mutex);//使用临界资源前需要加锁
    //临界区代码段...
    V(mutex);//使用临界资源后需要加锁
    //...
}

P2(){
```

```
    //...
    P(mutex);
    //临界区代码段...
    V(mutex);
    //...
}
```



信号量机制实现进程同步

进程同步：要让各并发进程按要求有序地推进。

比如，P1、P2并发执行，由于存在异步性，因此二者交替推进的次序是不确定的。

若P2的”代码4“要基于P1的”代码1“和”代码2“的运行结果才能执行，那么我们就必须保证”代码4“一定是在”代码2“之后才会执行。

这就是进程同步问题，让本来异步并发的进程互相配合，有序推进。

```
P1()
{
    //代码1;
    //代码2;
    //代码3;
}
P2()
{
    //代码4;
    //代码5;
    //代码6;
}
```

用信号量实现进程同步：

1. 分析什么地方需要实现”同步关系“，即必须保证”一前一后“执行的两个操作（或两句代码）
2. 设置同步信号量S，初始为0
3. 在”前操作“之后执行V(S)
4. 在”后操作“之前执行P(S)

   技巧口诀：前V后P

```
//信号量机制实现同步
semaphore S=0;//初始化同步信号量，初始值为0
P1()
```

```
{
    //代码1;
    //代码2;
    V(S);
    //代码3;
}
P2()
{
    P(S);
    //代码4;
    //代码5;
    //代码6;
}
//保证了代码4一定是在代码2之后执行
```

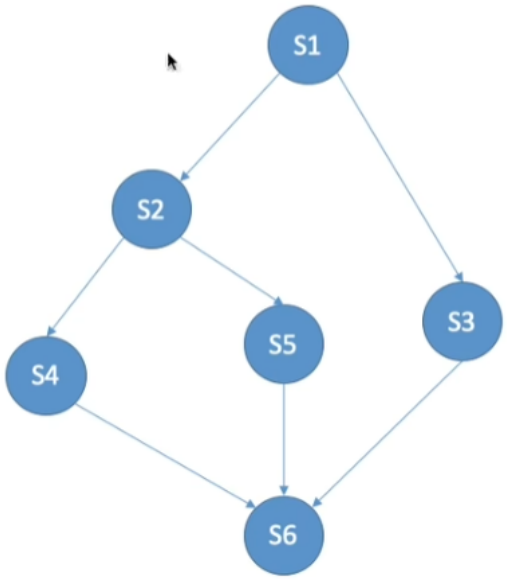
若先执行到V(S)操作，则S++后S=1。之后当执行到P(S)操作时，由于S=1，表示有可用资源，会执行S--，S的值变回0，P2进程不会执行block原语，而是继续往下执行代码4。

若先执行到P(S)操作，由于S=0，S--后S=-1，表示此时没有可用资源，因此P操作中会执行block原语，主动请求阻塞。之后当执行完代码2，继而执行V(S)操作，S++，使S变回0，由于此时有进程在该信号量对应的阻塞队列中，因此会在V操作中执行wakeup原语，唤醒P2进程。这样P2就可以继续执行代码4了

理解：信号量S代表”某种资源“，刚开始是没有这种资源的。P2需要使用这种资源，而又只能由P1产生这种资源

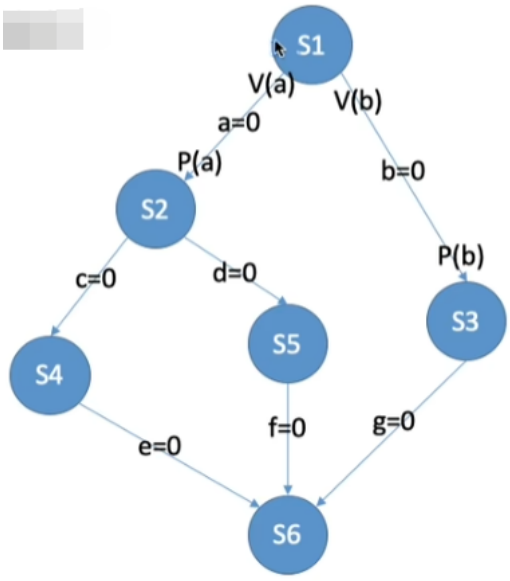
信号量机制实现前驱关系

进程P1中有句代码S1，P2中有句代码S2，P3中有句代码S3...P6中有句代码S6。这些代码按如下前驱图所示的顺序来执行：



其实每一对前驱关系都是一个进程同步问题（需要保证一前一后的操作）因此，

1. 要为每一对前驱关系设置一个同步信号量
2. 在”前操作“之后对相应的同步信号量执行V操作
3. 在”后操作“之前对相应的同步信号量执行P操作




```
P1() {
  ...
  S1;
  V(a);
  V(b);
  ...
}
```

```
P2() {
  ...
  P(a);
  S2;
  V(c);
  V(d);
  ...
}
```

```
P3() {
  ...
  P(b);
  S3;
  V(g);
  ...
}
```

```
P4(){
  ...
  P(c);
  S4;
  V(e);
  ...
}
```

```
P5(){
  ...
  P(d);
  S5;
  V(f);
  ...
}
```

```
P6() {
  ...
  P(e);
  P(f);
  P(g);
  S6;
  ...
}
```