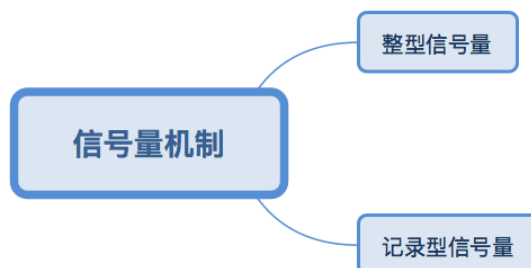


本节内容

信号量机制

王道考研/CSKAOYAN.COM

知识总览



复习回顾+思考：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

王道考研/CSKAOYAN.COM

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：系统中只有一台打印机，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“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)**

王道考研/CSKAOYAN.COM

信号量机制——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。
Eg：某计算机系统中有一台打印机...

```
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);
...
```

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

存在的问题：不满足“让权等待”
原则，会发生“忙等”

王道考研/CSKAOYAN.COM

信号量机制——记录型信号量

整型信号量的缺陷是存在“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) {
        block (S.L);
    }
}
```

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

```
/*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) {
        wakeup (S.L);
    }
}
```

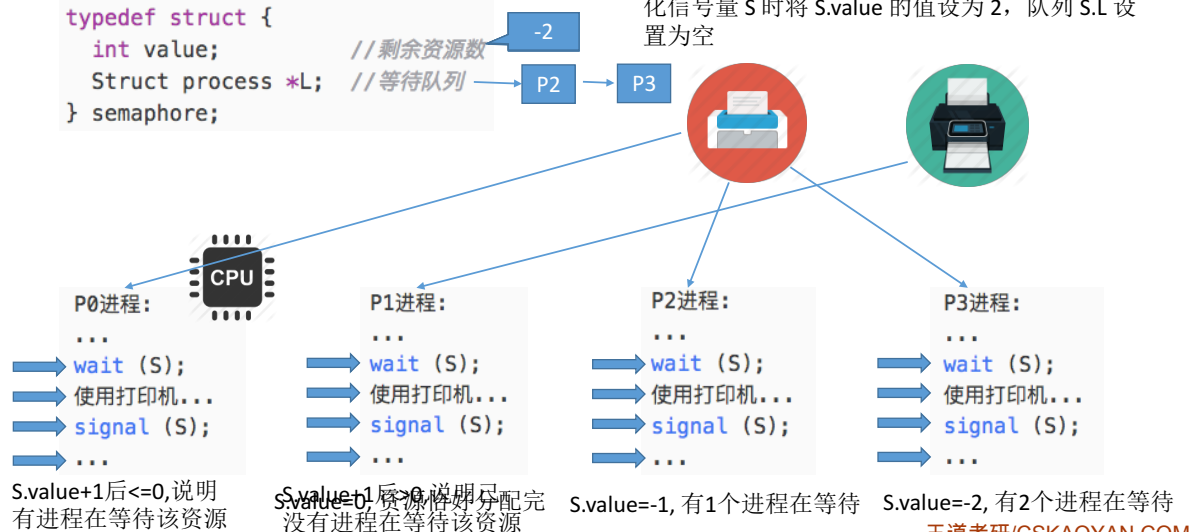
释放资源后，若还有别的进程在等待这种资源，则使用wakeup原语唤醒等待队列中的一个进程，该进程从阻塞态变为就绪态

王道考研/CSKAOYAN.COM

信号量机制——记录型信号量

```
/*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;
```

Eg: 某计算机系统中有2台打印机...，则可在初始化信号量S时将S.value的值设为2，队列S.L设置为空



王道考研/CSKAOYAN.COM

信号量机制——记录型信号量

```

/*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) {
        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) {
        wakeup(S.L);
    }
}

```

在考研题目中 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”。

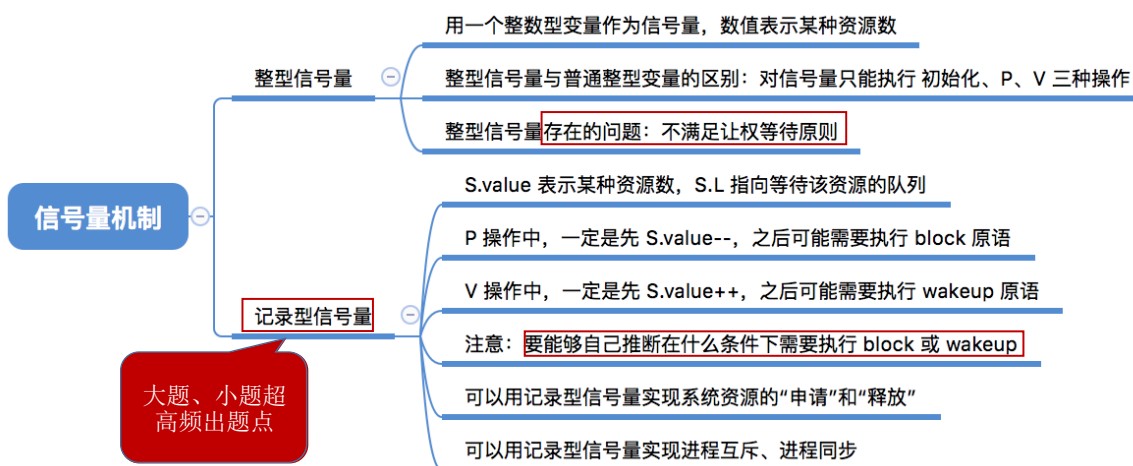
S.value 的初值表示系统中某种资源的数目。

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 原语唤醒等待队列中的第一个进程（被唤醒进程从阻塞态 → 就绪态）。

王道考研/CSKAOYAN.COM

知识回顾与重要考点



注：若考试中出现 P(S)、V(S) 的操作，除非特别说明，否则默认 S 为记录型信号量。

王道考研/CSKAOYAN.COM