第6章 进程同步

6.5 信号量集机制

6.5 信号量集机制-AND型信号量

- AND型信号量的基本思想是：
  - 将进程在整个运行过程中需要的多类资源，一次性地全部分配给进程，待该进程使用完后再一起释放。
  - 只要有一个资源未能分配给该进程，其他所有资源也不分配。

- 我们称AND型信号量的P原语为SP或Swait，V原语为SV或Ssignal。

SP操作

SP(S_1, S_2, …, S_n)

{ if(S_1 >= 1 & S_2 >= 1 & … & S_n >= 1)

for (i=1; i<= n; i++) S_i= S_i-1;

else

{将进程插入第一个小于1的信号量等待队列;

将调用进程的程序计数器置为SP的第一条指令;

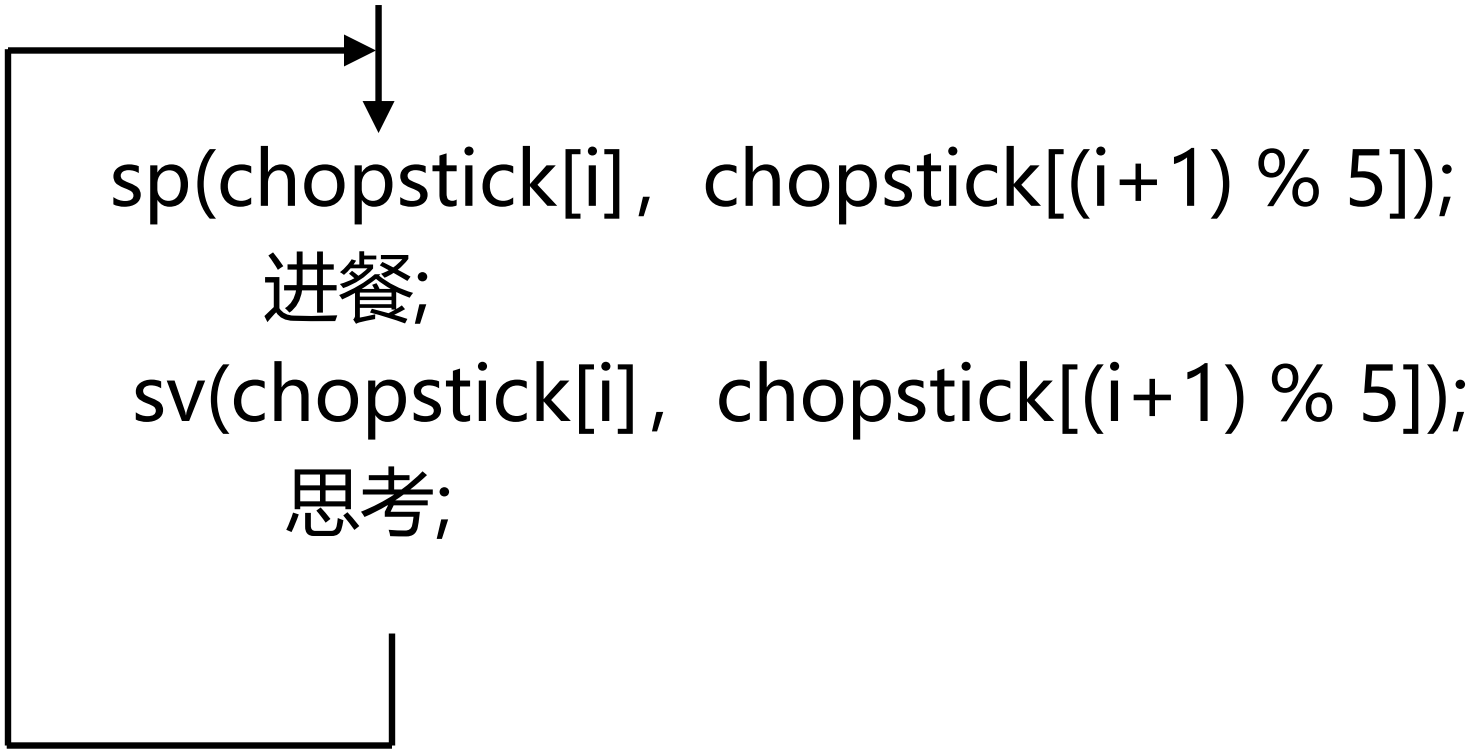
}

}

SV操作

SV(S_1，S_2，…，S_n)
{ for (i=1; i<=n; i++)
  { S_i= S_i+1;
    唤醒S_i等待队列上的所有进程，
    并将它们插入就绪队列;
  }
}

用AND型信号量解决哲学家进餐问题



一般信号量集

- 信号量集是AND型信号量的扩充，其基本思想是：
  - 在一次原语操作中完成对所有资源的申请
  - 即进程可以一次申请多类资源，每类资源可以申请多个
  - 当某类资源的数量低于其下限值或不能满足进程的申请要求时，则不进行分配。

SP

SP(S_1,t_1,d_1，S_2,t_2,d_2，…，S_n,t_n,d_n)

/*t_i为下限值，d_i为资源申请量*/

{

if (S_1>=t_1 && S_1>=d_1 && … && S_n>=t_n && S_n>=d_n)

for (i=1 ; i<=n; i++)

S_i=S_i - d_i;

else

{ 将进程插入第一个资源数小于t_i或d_i的信号量的等待队列;

将调用进程的程序计数器设置为SP的第一条指令;

}

}

SV

SV(S_1,d_1, S_2,d_2, …, S_n,d_n)

{

for(i=1; i<=n; i++)

{

S_i=S_i+ d_i;

唤醒队列S_i上的所有进程,

并将它们插入就绪队列;

}

}

信号量集的几种特殊的情况

- SP(S，d，d):此时信号量集中只有一个信号量，它每次申请d个资源，当资源数量少于d个时，便不予分配。
- SP(S，1，1)：此时的信号量集已退化为基本型信号量（记录型信号量）。
- SP(S，1，0)：这是一种很特殊的信号量，可作为一个可控开关。当S≥1时，允许多个进程进入特定区域;当S＝0时，禁止任何进程进入特定区。

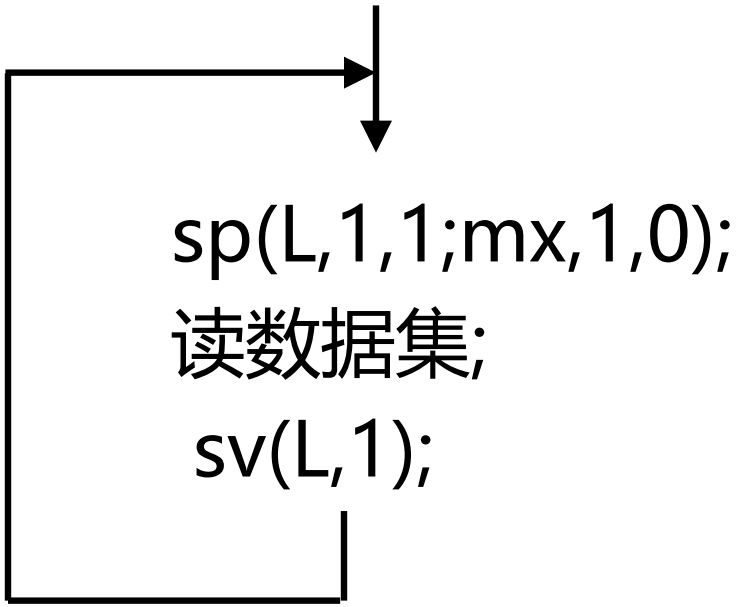
用信号量集解决读者-写者问题

- 为了实现方便增加一个限制条件，即最多只允许RN个读进程同时读。

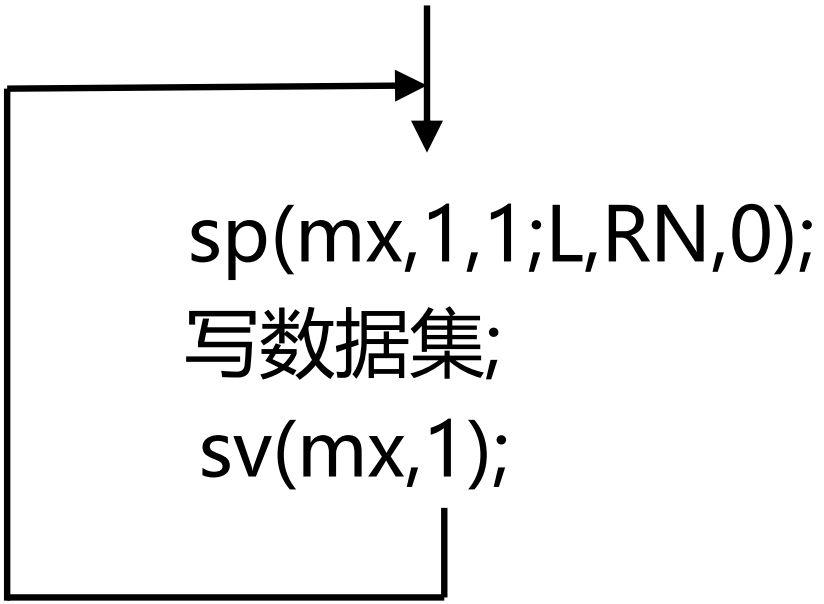
- 设置两个信号量：
  - mx表示写互斥的信号量，用于实现写进程与读进程的互斥以及写进程与写进程的互斥，其初值为1;
  - L表示允许读进程数目的信号量，用来说明系统还可以允许多少个读进程进入，其初值为RN。

算法描述

- 读者



- 写者



第6章 进程同步

6.6 管程

6.6 管程

- 信号量的同步操作分散在各进程中不便于管理，还可能导致系统死锁。如：生产者消费者问题中将P颠倒可能死锁。
- 为此Dijkstra于1971年提出：把所有进程对某一种临界资源的同步操作都集中起来，构成一个所谓的秘书进程。凡要访问该临界资源的进程，都需先报告秘书，由秘书来实现诸进程对同一临界资源的互斥使用。
- 1973年Hansen和Hoare又把秘书进程的思想发展为管程的概念。

有了管程的优势

- 把分散在各进程中的临界区集中起来进行管理；
- 防止进程有意或无意的违法同步操作，
- 便于用高级语言来书写程序，也便于程序正确性验证。

6.6.1 管程定义

- Hansen为管程(Monitor)所下的定义是：管程定义了一个数据结构和能为并发进程所执行的一组操作，这组操作能同步进程和改变管程中的数据。

  - 这些数据结构是对相应临界资源的抽象
  - 管程：代表临界资源的数据及在其上操作的一组过程

管程的构成

- 管程的名字
- 局部于管程的共享数据结构（变量）
- 对共享数据结构进行的一组操作（过程或函数）
- 对局部于管程的数据设置初始值的语句

管程的语法

```
Monitor  monitor_name;
variable  declarations;
void  Entry P1(...)
        { … }
void  Entry P2(...)
        { … }
        ¦
void Pn(...)
        { … }
{
    initialization code;
}
```

/*管程名*/

/*共享变量说明*/

/*对数据结构操作的函数*/

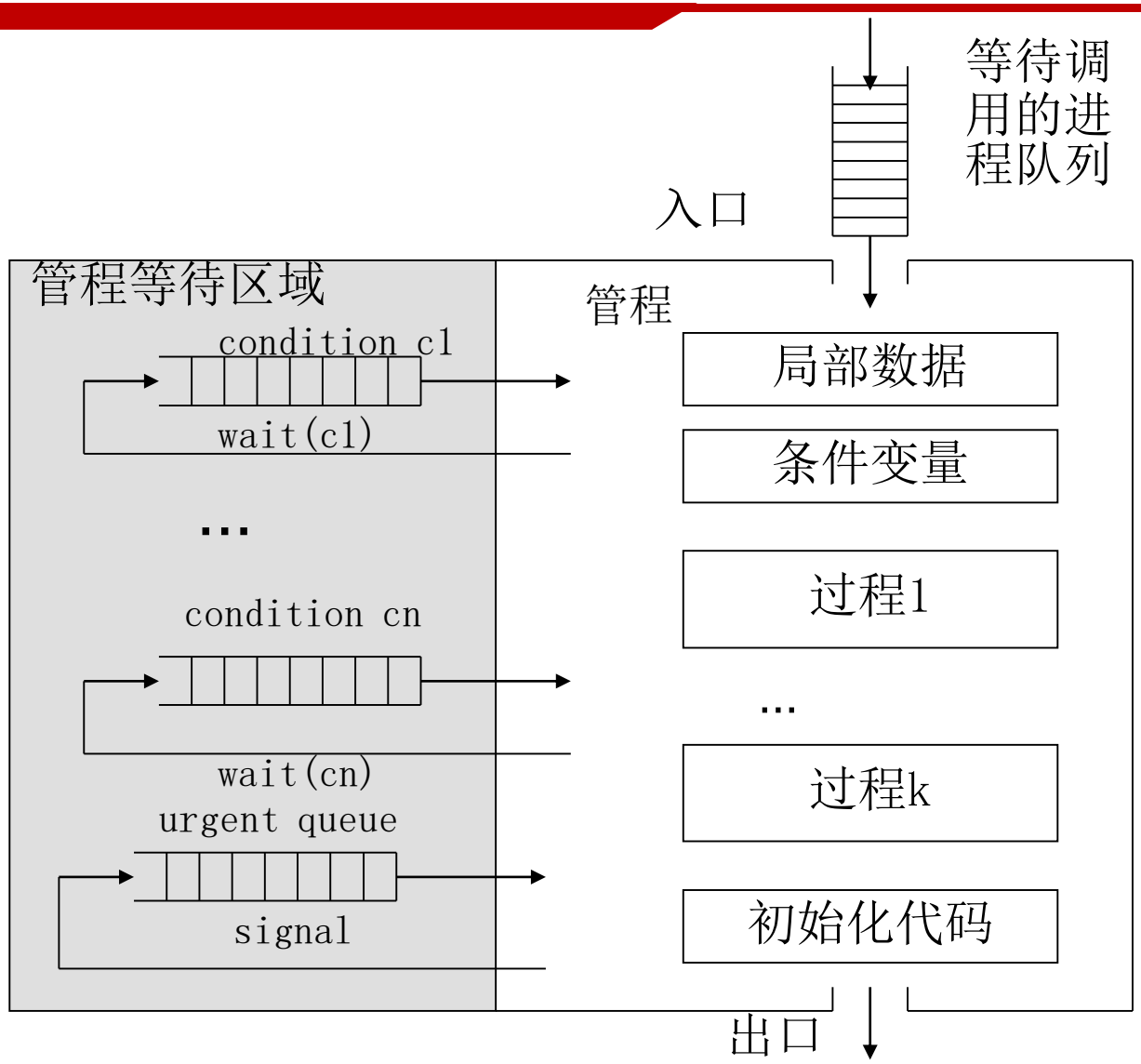
管程内的函数分外部函数和内部函数两种，外部函数可以被管程外的进程调用，内部函数只有管程内的函数才能调用。(外部函数在函数名前用entry标识)

/*设初值语句*/

管程的基本特性

1. 安全性
   - 管程内的局部数据**只能**被该管程内的函数所访问。
2. 共享性
   - 进程对管程的访问**都可以通过**共享相同的管程函数来实现。
3. 互斥性：
   - **每次仅允许一个**进程在管程内执行某个函数，共享资源的进程可以访问管程，但是**只有至多一个**调用者**真正**进入管程，其他调用者**必须**等待。
4. 透明性
   - 管程是一个语言成分，所以管程的互斥访问完全由编译程序在编译时自动添加上，**无需**程序员关心，而且保证正确。

管程的结构


等待调用的进程队列
入口
管程等待区域
condition c1
wait(c1)
...
condition cn
wait(cn)
urgent queue
signal
管程
局部数据
条件变量
过程1
...
过程k
初始化代码
出口

入口等待队列

- 因为管程是互斥进入的，所以当一个进程试图进入一个已被占用的管程时它应当在管程的入口处等待，因而在管程的入口处应当有一个进程等待队列，称作入口等待队列。

条件变量和条件队列

- 由于管程通常是用于管理资源的，因而在管程内部，应当存在某种等待机制。当进入管程的进程因资源被占用等原因不能继续运行时使其等待。为此在管程内部可以说明和使用一种特殊类型的变量——条件变量。
- 条件变量是当调用管程过程的进程无法运行时，用于阻塞进程的**一种特殊的信号量**。
- 每个条件变量表示一种等待原因，对应一个等待队列（条件队列）。

当一个进程通过调用管程的外部函数而进入管程之后，因为某种原因而无法运行时，就在相应的条件变量上等待。

条件变量（续）

- 条件变量用于区别各种不同的等待原因。其说明形式为： condition x;
- 还应设置在条件变量上进行操作的两个同步原语wait和signal。使用方式为：x.wait，x.signal。
  - x.wait 使调用进程等待，并将它排在x的队列上；
  - x.signal 将x队列的队首进程唤醒（如果x队列为空，相当于空操作，调用进程继续）。

条件变量（续）

- 注意：条件变量与P、V操作中信号量是不同的!
  - P、V操作内部会对信号量进行计数，通过对累计值的管理，实现对进程间的关系进行有效管理
  - 条件变量只是一种简单的信号量，只进行维护等待队列的操作，不进行计数，没有累计值。若条件变量上没有等待的进程，信号量会被丢弃，即signal触发一个空操作。

问题 - 多个进程出现在管程中

- 当一个进入管程的进程执行等待操作时，它应当释放管程的互斥权（允许入口等待队列上的一个进程进入）；
- 当进入管程的进程执行唤醒操作时（如 P 唤醒 Q），管程中便存在两个同时处于活动状态的进程。处理方法有三种：
  - P 等待 Q 继续，直到 Q 退出或等待 （Hoare采用）
  - Q 等待 P 继续，直到 P 退出或等待
  - 规定管程中的过程所执行的唤醒操作是过程体的最后一个操作 （Hansen采用）

问题（续）

- 按照Hoare的方法，如果进程 P 唤醒进程Q，则 P 等待Q继续，如果进程Q在执行又唤醒进程 R，则 Q等待 R 继续，……，如此，在管程内部，由于执行唤醒操作，可能会出现多个等待进程。
- 因而还需要有一个进程等待队列，这个等待队列被称为紧急等待队列。它的优先级应当高于入口等待队列的优先级。

问题（续）

- x.wait：如果紧急等待队列非空，则唤醒第一个等待者，否则释放管程的互斥权；执行此操作的进程将自己阻塞在x队列中。

- x.signal：如果x队列为空，则相当于空操作，执行此操作的进程继续；否则唤醒x队列的第一个等待者，执行此操作的进程排入紧急等待队列的尾部。

6.6.2 利用管程解决哲学家进餐问题

- 用三种不同状态表示哲学家的活动：进餐、饥饿、思考。

（thinking，hungry，eating）state[5];

- 为每个哲学家设置一个条件变量self（i），当哲学家饥饿又不能获得筷子时，用self来阻塞自己：

condition self[5];

- 管程设置三个函数：pickup取筷子，putdown放筷子，test测试是否具备进餐条件。

哲学家进餐问题的解法

```
monitor DiningPhilosophers
{
    enum { THINKING; HUNGRY, EATING) state [5] ;
    condition self [5];

    void pickup (int i) {
            state[i] = HUNGRY;
            test(i);
            if (state[i] != EATING) self[i].wait;
}

   void putdown (int i) {
            state[i] = THINKING;
           // test left and right neighbors
             test((i + 4) % 5);
             test((i + 1) % 5);
}
```

哲学家进餐问题的解法

```
    void test (int i) {
            if ((state[(i + 4) % 5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i + 1) % 5] != EATING) ) {
                state[i] = EATING ;
               self[i].signal () ;
            }
   }

      initialization_code() {
          for (int i = 0; i < 5; i++)
          state[i] = THINKING;
        }
}
```

哲学家进餐问题的解法

```
main()
{
   cobegin
      philosopher(0);
      philosopher(1);
      philosopher(2);
      philosopher(3);
      philosopher(4);
   coend
}
```

```
void philosopher(int i)
{
    while(true)
    {
        Thinking;
         DiningPhilosophers.pickup(i);
         Eating;
         DiningPhilosophers.putdown(i);
     }
}
```

本章小节

- 临界资源访问的原则
- 同步与互斥的定义
- 互斥的实现方法（软件、硬件、锁）
- 信号量的定义
- 信号量解决经典问题
- 信号量集的基本概念
- 管程的基本概念

小作业

1. 写出无死锁的哲学家进餐问题的信号量解法。
2. 有P1、P2、P3三类进程（每类进程都有K个）共享一组表格F（F有N个），P1对F只读不写，P2对F只写不读，P3对F先读后写。进程可同时读同一个Fi（0≤i ≤N）；但有进程写时，其他进程不能读和写。
   - 用信号量和P、V操作给出方案
   - 对方案的正确性进行分析说明
   - 对访问的公平性进行分析

小作业

3. 独木桥问题

- 独木桥只允许一台汽车过河，当车到达桥头时，如果桥上无车，则可上桥，否则车在桥头等待，直到桥上无车。使用PV操作解决该问题。
- 如果独木桥允许多台车同方向过河，当车到达桥头时，如果桥上只有同方向车且不超过N台，或者桥上无车，则可上桥通过，否则等待，直到满足上桥条件。使用PV操作解决该问题。

4. 红黑客过河问题

- 有红客和黑客两组人员需要过河。河上有船，但是每次只能乘坐4人，且每次乘客满员时才能开船，到河对岸后空船返回。但船上乘客不能同时有3个红客 、1个黑客 或者 1个红客 、 3个黑客的组合。（其他组合安全）。请用PV操作解决红客、黑客过河的问题。

大作业

1. V9版，课后编程项目，针对读者/写者、生产者/消费者、哲学家问题、理发师（or睡觉助教），选择一个实现，要求使用Pthreads 和 Windows 线程API实现