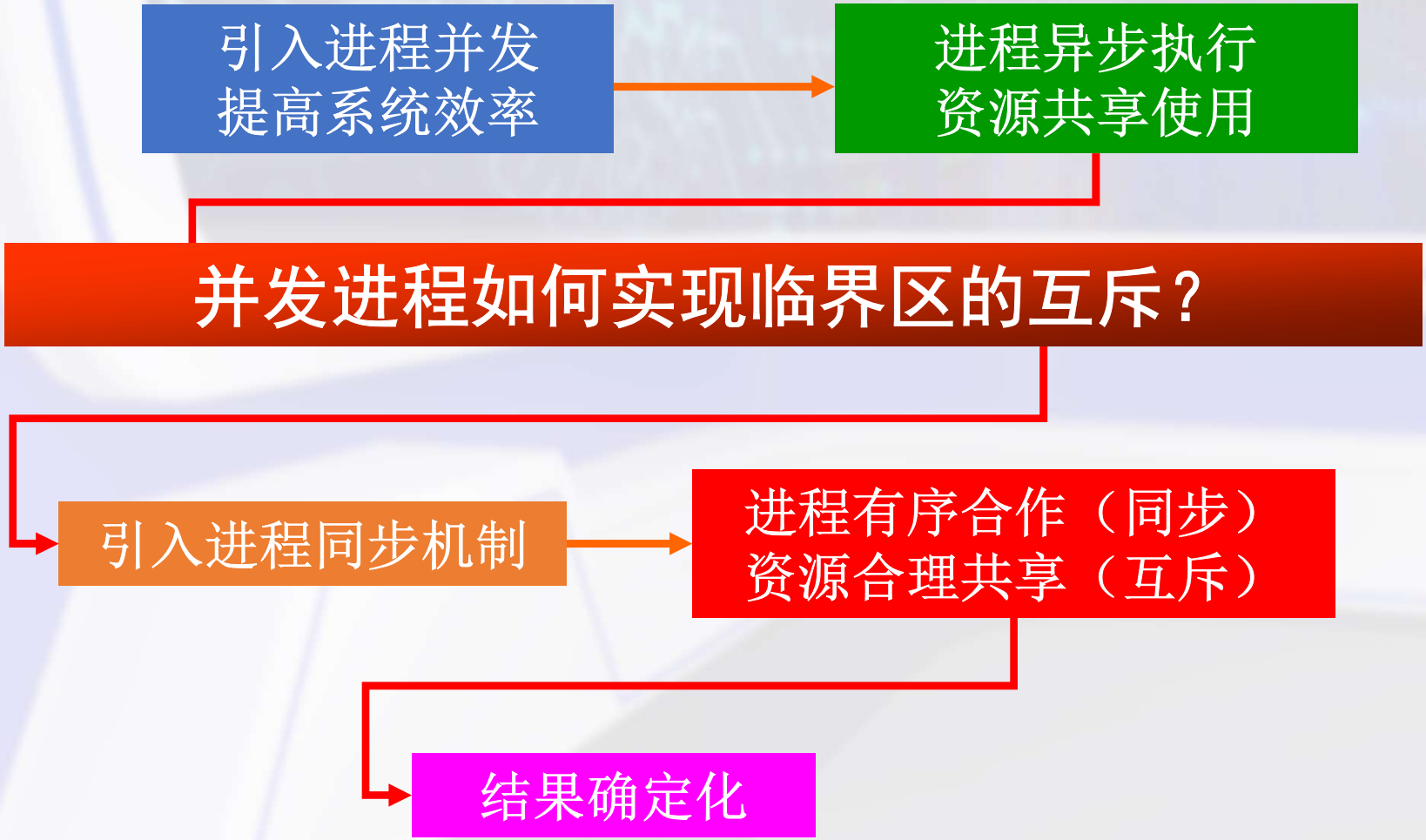
临界区管理

Linux
Android
Linux
OpenStack
Mac OS
Windows

上节课的重点内容


引入进程并发
提高系统效率
进程异步执行
资源共享使用
并发进程如何实现临界区的互斥？
引入进程同步机制
进程有序合作（同步）
资源合理共享（互斥）
结果确定化

临界区管理

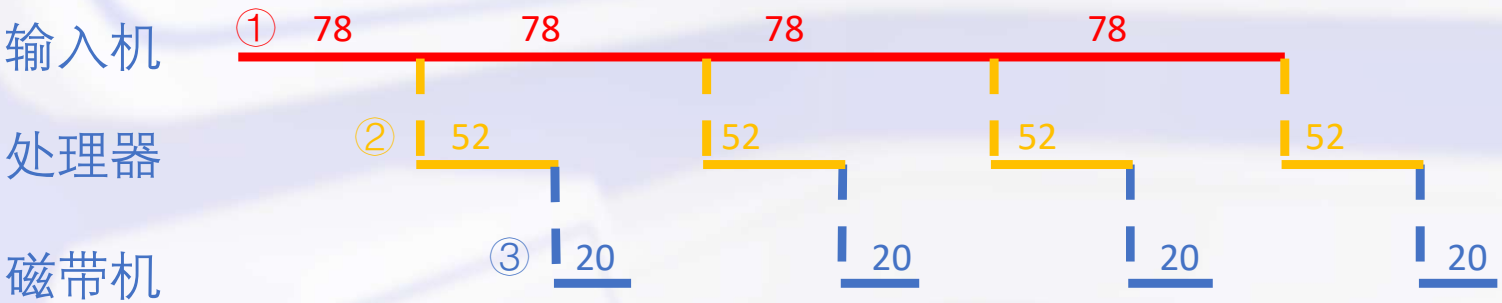
本讲内容

1. 临界区及其使用原则
2. 临界区管理软件方法
3. 临界区管理硬件方法
4. 软硬件方法的优缺点

进 程 联 系

并发程序

把一个具体问题求解设计成若干个可同时执行的程序模块的方法，提高了计算效率



进 程 联 系

进程同步

并发进程之间为完成共同任务，基于某个条件来协调执行先后关系而产生的协作制约关系

进程互斥

并发进程之间因互相争夺独占性资源而产生的竞争制约关系

互斥与临界区

临界资源　多个进程使用的互斥共享变量所代表的资源

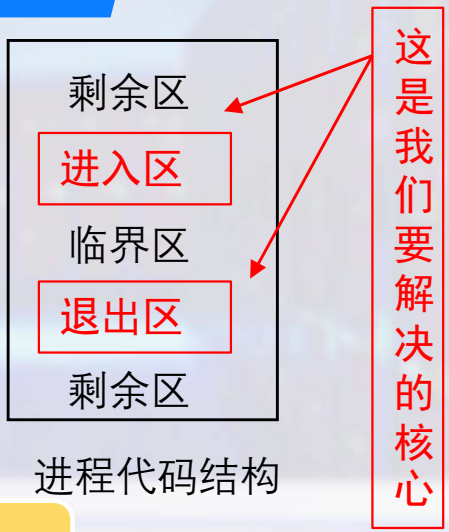
即一次只能被一个进程使用的资源

临界区　并发进程中与互斥共享变量相关的程序段

- 多个并发进程访问临界资源时，存在竞争制约关系
- 如果两个进程同时停留在相关的临界区内，就会出现与时间相关的错误

与时间有关的不确定

例 P1和P2是并发的终端订票进程，x是票数



进程代码结构

P1	P2
① Read(x);	④ Read(x)
②if x>=1 then x:=x-1	⑤if x>=1 then x:=x-1
③ write(x)	⑥ write(x)

这就是临界区，一次只允许一个进程进入的该段进程的那一段代码，中断导致卖出同一张票

临界区管理的三个要求

- 基本原则：互斥进入：如果一个进程在临界区中执行，则其他进程不允许进入，一次至多允许一个进程停留在相关的临界区内
- 这些进程间的约束关系称为互斥，这保证了临界区的正确性

好的临界区保护原则

- 有空让进：当临界区空闲的时候，至少可以允许一个进程进入临界区
- 有限等待：一个进程不能无限制地等待进入临界区 -“饥饿”
- 让权等待：当进程不能进入临界区时，应立即释放处理机，防止进程忙等待

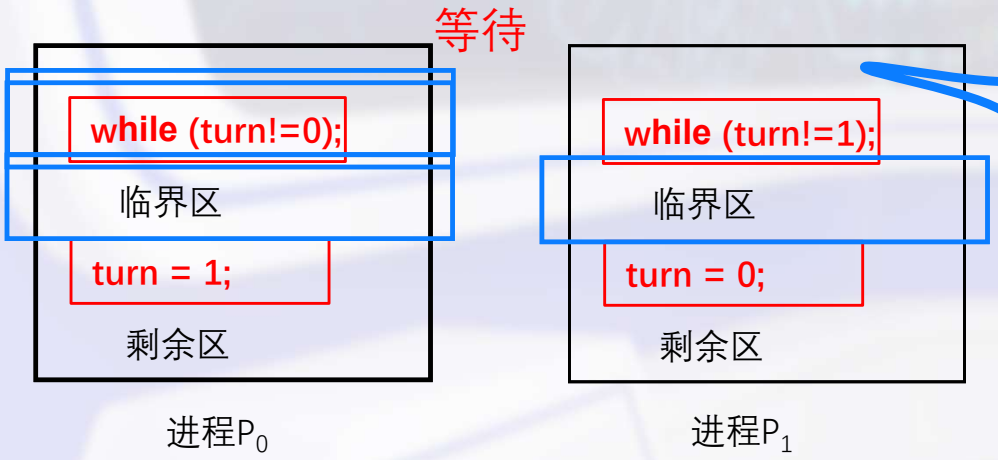
临界区管理

本讲内容

1. 临界区及其使用原则
2. 临界区管理软件方法
3. 临界区管理硬件方法
4. 软硬件方法的优缺点

临界区管理失败尝试

进入临界区的一种失败尝试 – 轮换法/值日

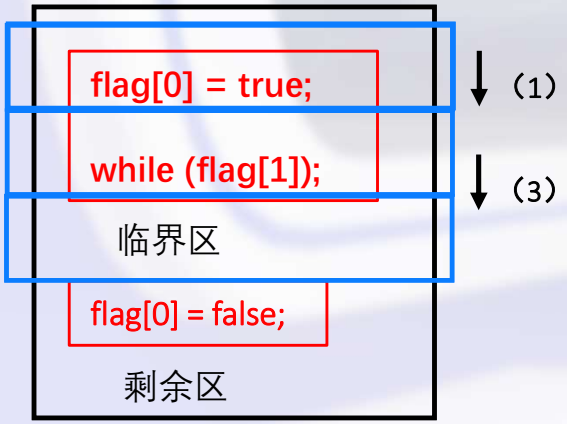


- 轮到P_1但被中断，造成P_0没法进入
- 问题：[illegible]进程P[illegible]在临界区，P_0完成后[illegible]能[illegible]看再次进入（不满足有空让进）

- 进入之前检测是否轮到自己，运行结束后让给对方
- 如果两个进程都进入临界区，则turn即=0又等于1，存在矛盾（满足互斥进入）

临界区管理失败尝试

进入临界区的一种失败尝试 – 举旗法



(1)

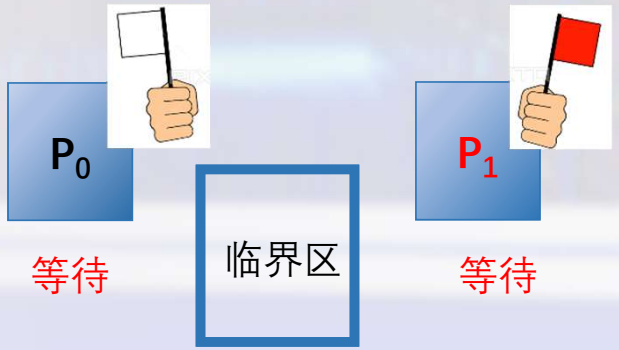
(3)

进程P_0

(2)

(4)

进程P_1



- 此时进程P_0和P_1都请求进入，造成卡死无限等待（不满足有空让进）
- 进入之前检测对方是否想进入（满足互斥进入）
- 如果两个进程都进入临界区，则flag[1]即=true又等于false，存在矛盾
- 能够实现：某一进程多次执行，而另一线程不执行

临界区管理软件方法

Peterson算法 - 结合轮换和举旗法

```
flag[0] = true
turn = 1 ;
while (flag[1]&&turn== 1);
    临界区
flag[0] = false;
    剩余区
```

进程P_0

```
flag[1] = true;
turn = 0;
while (flag[0]&&turn == 0);
    临界区
flag[1] = false;
    剩余区
```

进程P_1

- 轮流轮到但是不想进的时候，就留给想进入举旗的人
- 当大家都举旗想进入的时候，根据轮流原则进入

临界区管理软件方法

Peterson算法思想

- 想进入临界区之前，P_0 / P_1都要举起自己的旗子
  - P_0 确认旗子举好以后，往临界区门上贴“轮到 P_1 使用”
  - P_1 确认旗子举好以后，往临界区门上贴“轮到 P_0 使用”
- 如果对方的旗子没有举起来，则可以进入临界区

- 如果对方的旗子举起来，且不轮到自己访问（门上的名字不是自己）则等待，否则可以进入临界区
- 出临界区后，放下自己的旗子

```
flag[ i ] = true;
turn = j;
while ( flag[ j ] && turn==j );
    临界区
flag[ j ] = false;
    剩余区
```

进程P_i

临界区管理软件方法

Peterson算法思想

- 进入临界区的情况
  - 如果只有一个人举旗（flag[j]=flase）就可以直接进入
  - 如果两个人同时举旗，由临界区门上的名字决定谁进
    - 后来的人把临界区的门贴成先到的人
    - 手快有（被另一个人的名字覆盖）、手慢無

```
flag[ i ] = true;
turn = j;
while ( flag[ j ] && turn==j );
    临界区
flag[ j ] = false;
    剩余区
```

进程P_i

课后习题

题

Peterson算法如何扩展到多进程?

- Peterson算法只是解决了两个进程的竞争问题，那么解决多个进程竞争的完全互斥算法又是怎样的？有没有什么可行的解决方案？

1. 进程是系统进行资源分配、调度和保护的独立单位。操作系统为管理进程而设置的与进程一一对应的数据结构为进程控制块/PCB，它包含进程描述信息、进程控制信息、所拥有的资源和使用情况和CPU现场保护信息。
2. 处理器调度中，决定进程由挂起就绪态向就绪态转换的调度是中级调度，决定进程由就绪态向运行态转换的调度是低级/进程调度。

3、在Linux系统中，创建进程的原语是fork。

4、使用临界区一般应该遵循的原则包括：有空让进、无空等待、多中择一、有限等待和让权等待。（答案顺序任意）

1. 下列关于并发和并行的说法错误的是D。
   A．并发是指多个事件在同一时间间隔内发生
   B．并行是指多个事件在同一时刻发生
   C．在单机系统内进程只能并发执行，不能并行
   D．并发和并行是对同一现象的两种不同称呼，本质是一样的

2、某进程由于时间片用完退出CPU后其状态将从运行态转换为 B 。

A．等待态　　B．就绪态

C．新建态　　D．终止态

3、（ A ）是指作业从提交给系统到完成的时间间隔

A）周转时间　　B）响应时间

C）等待时间　　D）运行时间

4、下列调度算法中，根据作业估计运行时间进行选择的是（ B ）。

A）先来先服务　　B）短作业优先

C）均衡　　D）最高响应比优先

5、一个进程被唤醒意味着（ D ）。

A）该进程重新占有了CPU　　B）该进程优先权变为最大

C）其PCB移至等待队首队列　　D）进程变为就绪态

临界区管理

本讲内容

1. 临界区及其使用原则
2. 临界区管理软件方法
3. 临界区管理硬件方法
4. 软硬件方法的优缺点

临界区管理硬件方法

1 测试并建立指令TS

TS: Test测试并Set建立，用硬件指令，实现对临界区代码的互斥执行

s代表临界资源状态，由TS指令控制

```
s : boolean; s := true;
process Pi        /* i = 1,2,…,n */
    pi : boolean;
begin
    repeat pi := TS(s)  until  pi;
    临界区;
    s := true;
end;
```

```
boolean TS(x)
   y : boolean
   begin
      y = x;
      x = false;
      return y;
   end
```

1. 将 x 设为 false
2. 返回旧 x 值

$x = True \rightarrow False$	$\mathrm{TS}(x) = True$
$x = False$	$\mathrm{TS}(x) = False$

临界区管理硬件方法

1 测试并建立指令TS

$x = True \rightarrow False$	$\mathrm{TS}(x) = True$
$x = False$	$\mathrm{TS}(x) = False$

s代表临界资源状态，由TS指令控制

```
s : boolean;   s := true;
process Pi        /* i = 1,2,…,n */
    pi : boolean;
begin
    repeat pi := TS(s)  until  pi;
    临界区;
    s := true;
end;
```

P1 s=true; pi= TS(true)=true; s=false;
:
跳出循环，P1进入临界区

P2: s=false; pi= TS(false)=false; s=false
P2…Pn等循环等待进入临界区

P2: s=true; pi= TS(true)= true; s=false;
P2进入临界区

代码简洁

临界区管理硬件方法

2 交换指令SWAP

- 交换指令将交换两个字的内容
- 公共变量lock决定临界区是否上锁
- 每个进程的私有变量key用于与lock交换

```
void SWAP(int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

进程P1

```
key = true; lock=false 初始不上锁
do {
    SWAP(&lock, key);
} while(key);      key = false; lock= true
临界区
lock := false;
```

临界区管理硬件方法

2 交换指令SWAP

- 交换指令将交换两个字的内容
- 公共变量lock决定临界区是否上锁
- 每个进程的私有变量key用于与lock交换

```
void SWAP(int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

进程P2

```
key = true;lock= true
do {
    SWAP(&lock, key);
} while(key);
临界区
lock := false;
```

key = true; lock= true

3 中断屏蔽方法

利用“开/关中断指令”实现（与原语的实现思想相同，即在某进程开始访问临界区到结束访问为止都不允许被中断，也就不能发生进程切换，因此也不可能发生两个同时访问临界区的情况）

```
...
关中断;
临界区;
开中断;
...
```

关中断后即不允许当前进程被中断，也必然不会发生进程切换

直到当前进程访问完临界区，再执行开中断指令，才有可能有别的进程上处理机并访问临界区

把并发环境改变成顺序执行环境

优点：简单、高效

缺点：不适用于多处理机；只适用于操作系统内核进程，不适用于用户进程（因为开/关中断指令只能运行在内核态，这组指令如果能让用户随意使用会很危险）

临界区管理

本讲内容

1. 临界区及其使用原则
2. 临界区管理软件方法
3. 临界区管理硬件方法
4. 软硬件方法的优缺点

软硬件方法的优缺点

- 软硬件方法都采用了忙等待方式
- 以上所有的解决方案都违反“让权等待”
- 让权等待：当进程不能进入临界区时，应立即释放处理机，防止进程忙等待

```
do {
    entry section;      //进入区
    critical section;   //临界区
    exit section;       //退出区
    remainder section;  //剩余区
} while(true)
```

```
acquire()
    while(!available)
        ;                       //忙等待
    available = false;          //获得锁
}
```

```
release(){
    available = true;           //释放锁
}
```

软硬件方法的优缺点

- 优点：等待期间不用切换进程上下文，多处理器系统重，若上锁时间短，则等待代价低
- 常用于多处理器系统，一个核忙等，其他核照常工作，并快速释放临界区　P1忙等只消耗一个核，竞争者P2可以同时执行
- 等不适用于单处理系统，忙等的过程中不可能解锁

P1忙等消耗一个核，单竞争者不可以同时执行，无法释放

软硬件方法的优缺点

1. 软硬件方法都采用了忙等待方式

2. 软件方法实现复杂，需要编程技巧

3. 硬件指令方法代码简洁有效

4. 硬件中断屏蔽方法代价较高

临界区管理

Linux
Android
Linux
OpenStack
Mac OS
Windows