3.2 临界区管理

- 互斥与临界区
- 临界区管理的尝试
- 软件算法
- 硬件设施

- 互斥与临界区
- 临界区管理的尝试
- 软件算法
- 硬件设施

3.2.1 互斥与临界区

- 飞机票售票系统之所以会产生错误，原因在于多个售票进程交叉访问了共享变量 Aj
- 并发进程中与共享变量有关的程序段叫“临界区”(Critical Section)，共享变量代表的资源叫“临界资源”(Critical Resource)

3.2.1 互斥与临界区（续）

- 与同一变量有关的临界区分散在各进程的程序段中，而各进程的执行速度不可预知
- 如果能保证进程在临界区执行时，不让另一个进程进入临界区，即各进程对共享变量的访问是互斥的，就不会造成与时间有关的错误

3.2.1 互斥与临界区（续）

- Dijkstra在1965年首先提出临界区的概念。可以用与一个共享变量相关的临界区的语句结构来书写交互的并发进程

```
shared  variable
region  variable  do  statement
```

- 临界区的嵌套使用

```
region x do [ … region y do [ … ] …]
region y do [ … region x do [ … ] …]
```

- 一个进程执行到临界区的语句时，不管该进程目前是否正在运行，都说它是在临界区内

3.2.1：临界区的调度原则

- 一次至多允许一个进程进入临界区内；
- 如果已有进程在临界区中，试图进入此临界区的其他进程应等待；
- 进入临界区的进程应在有限时间内退出，以便让等待队列中的一个进程进入
- 临界区调度原则可总结成四句话：
  1. 空闲让进：临界资源空闲时一定要让进程进入，不发生“互斥礼让”行为。
  2. 忙则等待：临界资源正在使用时外面的进程等待。
  3. 有限等待：进程等待进入临界区的时间是有限的，不会发生“饿死”的情况。
  4. 让权等待：进程等待进入临界区时应该放弃CPU的使用。

- 互斥与临界区
- 临界区管理的尝试
- 软件算法
- 硬件设施

3.2.2 临界区管理的尝试

- 为在具有一个处理器或共享主存的多处理器上执行的并发进程实现的。方法假定对主存中同一个单元的同时访问必定由存储器进行仲裁，使其串行化

3.2.2：尝试一

inside1= false
inside2= false

	process P1	process P2
1.	while inside2 do begin end;	
2.		while inside1 do begin end;
3.	inside1 := true;	
4.		inside2 = true;
5.	临界区;	
6.		临界区;
7.	inside1 := false;	
8.		inside2 := false;
9.	end;	end;

进程P1、P2同时进入了临界区

3.2.2：尝试一

inside1= false
inside2= false

```
      process P1                          process P2
1.    while inside2 do begin end;
2.
3.    inside1 := true;
4.
5.    临界区;
6.
7.    inside1 := false;
8.
9.    end;
```

1. 忙等
2. 若进程在临界区内失败且相应的inside为true，则其他进程将永久阻塞
3. 不能保证进程互斥进入临界区

进程P1、P2同时进入了临界区

3.2.2：尝试二

全局共享变量：

inside1= false

inside2= false

process P1	process P2
1. inside1 := true;	
2.	inside2 := true;
3. while inside2 do begin end;//无限等待	
4.	while inside1 do begin end;
5. /*临界区*/	/*临界区*/
6. Inside1:= false;	
7.	Inside2 := false;

- 互斥与临界区
- 临界区管理的尝试
- 软件算法
- 硬件设施

3.2.3 Perterson 算法

inside0= false; inside1= false
enum{0,1} turn;

	process P0	process P1
1.	inside0 = true;	inside1 = true;
2.	turn =1;	turn = 0;
3.	while (inside1 && turn == 1);	while (inside0 && turn==0);
4.	{ 临界区;}	{临界区;}
5.	inside0 = false;	inside1=false;
6.	end;	end;

进程P0只可能在while (inside[1] and turn=1) 处等待，在这里等待时，必定满足条件：inside[1]= true且turn=1且inside[0]= true

进程P1只可能在while (inside[0] and turn=0) 处等待，在这里等待时，必定满足条件：inside[0]= true且turn=0且inside[1]= true

上述两组条件是互斥的（turn值不可能同时既为0又为1），不可能同时成立，因而两个进程P0、P1决不会同时等待，进入死锁状态

- 互斥与临界区
- 临界区管理的尝试
- 软件算法
- 硬件设施

3.2.4：硬件设施

- 使用软件方法实现进程互斥使用临界资源是很困难的，他们通常能实现两个进程之间的互斥，很难控制多个进程的互斥
- 程序设计时应该非常注意，否则就会产生死锁和不能解决互斥
- 软件方法始终不能解决忙等，系统效率不高
- 用来实现互斥的硬件设施主要有：关中断、测试并建立指令、对换指令

3.2.4：关中断

- 进程切换需要依赖中断来实现，如果屏蔽中断，则不会引起进程切换
- 因此为了实现对临界资源的互斥使用，可以在进程进入临界区之前关闭中断，等进程退出临界区以后再打开中断
- 中断被屏蔽后，处理器不响应中断，则不会被切换
- 于是一旦屏蔽中断，进程就可以进入临界区修改访问共享变量，而不用担心其他进程介入

3.2.4：关中断

- 关中断是实现互斥的最简单方法之一
- 进程在测试标志之前，首先关中断，直到测试完并设置标志之后才开中断
- 进程在临界区执行期间，计算机系统不响应中断
- 因此，不会转向调度，也就不会引起进程或线程切换，正在执行标志测试和设置的进程或线程不会被打断，从而保证了互斥

3.2.4：关中断

- 关中断是实现互斥的最简单方法之一
- 进程在测试标志之前，首先关中断，直到测试完并设置标志之后才开中断
- 进程在临界区执行
- 因此，不会转向调
，正在执行标志测试从而保证了互斥

- 关中断时间过长会影响系统效率，限制处理器交叉执行程序的能力
- 关中断方法也不适用于多CPU系统，因为，在一个处理器上关中断，并不能防止进程在其他处理器上执行相同临界区代码

3.2.4：测试并建立指令

- 系统利用TS（Test and Set）指令实现临界区的互斥锁原语操作。
  - TS(x)： 若x=true，则x:=false；return true；否则 return false；
  - TS指令管理临界区时，可把一个临界区与一个布尔变量s相连，由于变量s代表了临界资源的状态，可把它看成一把锁

3.2.4：对换指令

- 对换（Swap）指令的功能是交换两个字的内容，处理过程如下：

```
Swap (a,b):
    temp:=a;
    a:=b;
    b:=temp;
```

- 在Intel 80x86中，对换指令称为XCHG指令

3.2.4：对换指令（例）

```
bool lock = false;
cobegin
  process Pi(){
  bool keyi = true;
  do {
    SWAP(keyi, lock);
  } while(keyi)          //上锁
  {临界区}
  SWAP (keyi, lock) ; //开锁
}
coend
```

lock == false:
无进程在临界区内

3.2.4：硬件设施（续）

- 忙等现象仍然存在，但由于使用机器指令测试，则忙等时间可以忍受
- 导致饥饿：临界区空闲时，执行循环检测的若干进程能进入临界区的几率是相等的，有的进程可能运气很不好，很难有机会进入临界区，因而饥饿
- 导致死锁：进程P1和P2，P1的优先级低于P2，P1通过测试进入临界区，这时P2到， P2抢占了P1的处理器开始执行， P2也要求进入临界区，但临界区被P1占，而P1的优先级没有P2高不可能抢占P2的处理器。这样，这两个进程互相等待， P1等P2释放处理器， P2一直忙等P1释放临界区，如果没有外力，这两个进程处于死锁

3.2：软件与硬件管理临界区

- 软件方法和硬件方法都存在忙等问题，浪费了处理器的时间
- 信号量方法可以实现进程同步与互斥，而不需要忙等