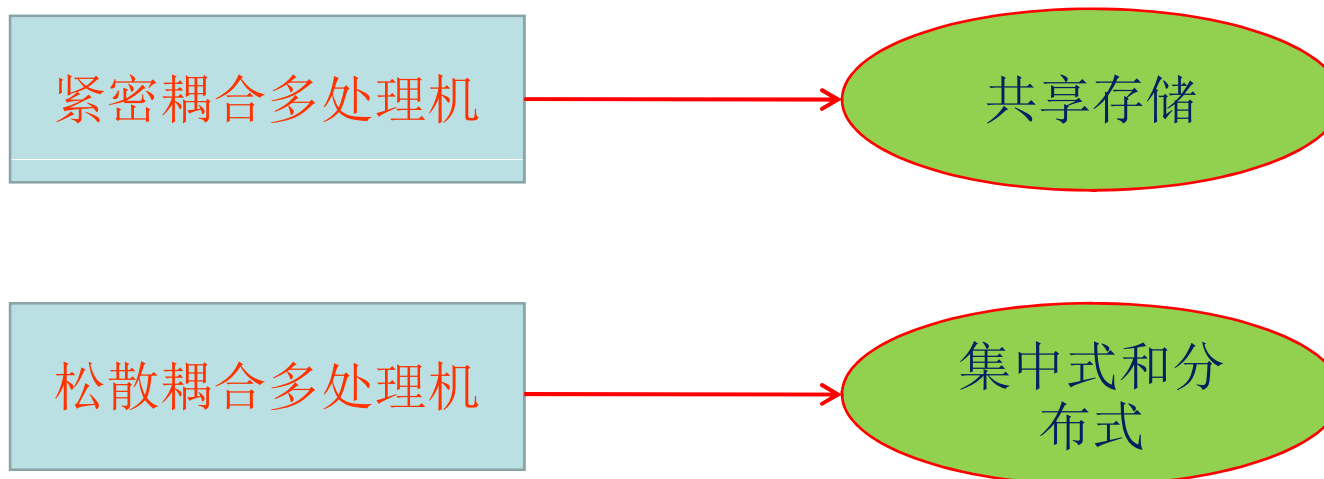


8.4 进程同步

多处理机系统中的进程间的同步显得更加重要和复杂。





8.4.1 集中式与分布式同步方式

1. 中心同步实体

为实现进程之间的同步，系统中必须有相应的同步实体(Synchronizing Entity)，如硬件锁、信号量以及进程等。如果该同步实体满足下述两个条件，则称之为中心同步实体：

- (1) 具有唯一的名字，并且为彼此必须同步的所有进程所知道。
- (2) 在任何时刻，这些进程中的任何一个都可以访问该同步实体。



2. 集中式同步机构

基于中心同步实体所构成的所有同步机构被称为集中式同步机构。相应的，其它同步机构则称为非集中式同步机构。

同步机制：

单处理机系统-----硬件锁，信号量

多处理机系统-----自旋锁，RCU锁，时间戳戳，事件计数
及中心进程。

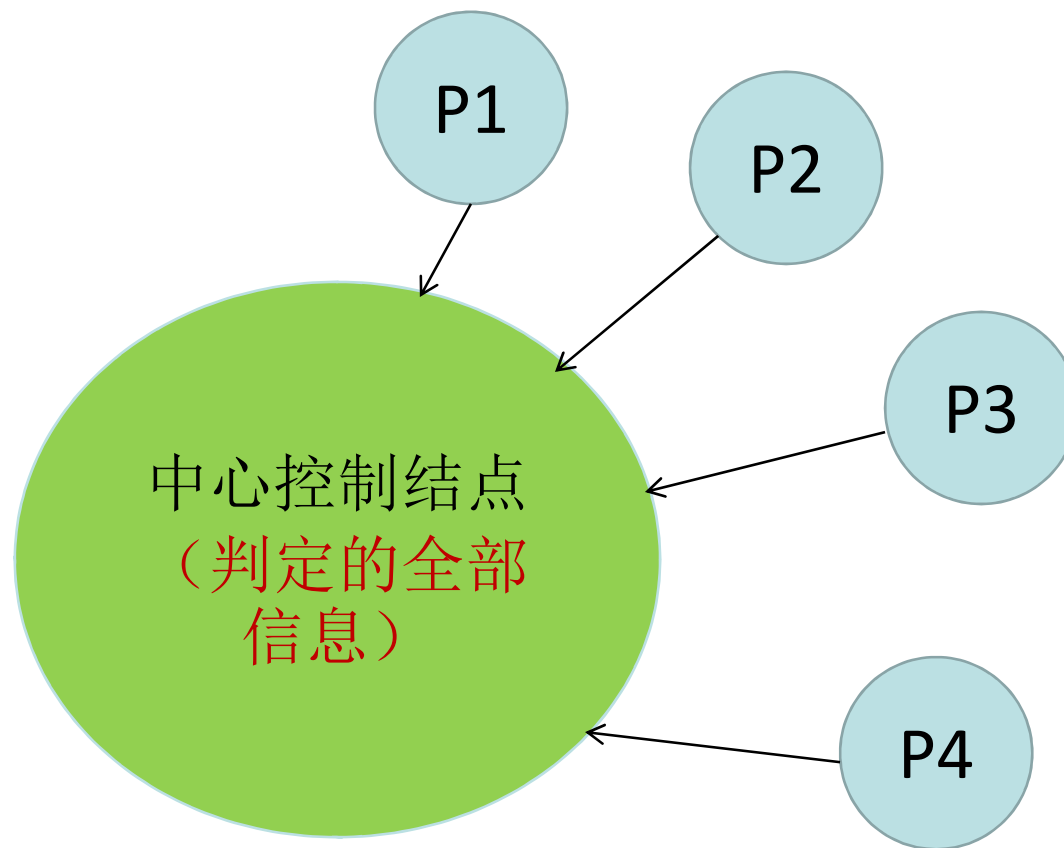
3. 集中式与分布式同步算法

在多台处理机系统中支持同步机构的相应同步算法，一般分为两种：

(1) 集中式同步算法

缺点

- ①可靠性差；
- ②易形成瓶颈。



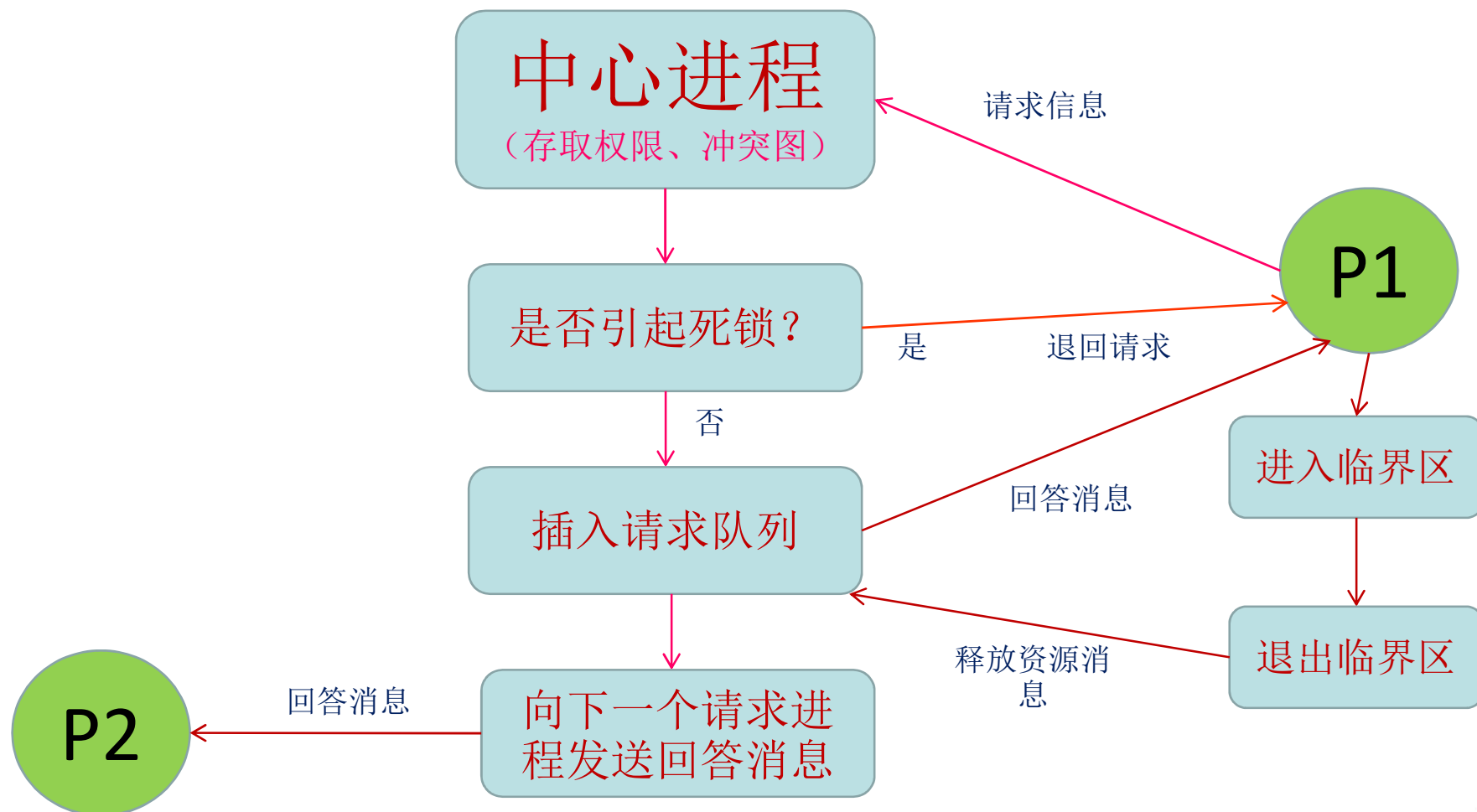


(2) 分布式同步算法。

- ①所有结点具有相同的信息；
- ②所有结点仅基于本地信息做出判断；
- ③为了做出最后的判定，所有的结点担负相同的职责；
- ④为了做出最后的判定，所有的结点要付出同样的工作量；
- ⑤一个结点发生故障，不会导致整个系统的崩溃。

请求、回答、释放

4. 中心进程方式





单处理机系统、共享存储器的多处理机系统

---集中式同步方式

分布式系统

---分布式同步方式

松散耦合式多处理机系统

---部分集中式，部分分布式



8.4.2 自旋锁 (spin lock)

1. 自旋锁的引入

在单CPU系统中，CPU在执行**读—修改—写**原语操作时，是具有原子性的，即在执行这些操作时不会被中断。保证原子性的基本方法是，在执行原语之前**关中断**，完成后再**开中断**。但是在对称多处理机系统中，这种原子性无法得到保证。

在多处理机系统中，还必须引入对**总线实现互斥**的机制。于是，**自旋锁机制**也就应运而生，并已大量应用于对总线资源的竞争。**自旋锁机制**并不局限于对总线资源的竞争。



2. 实现对总线互斥访问的方法

在总线上设置一个**自旋锁**，该锁最多只能被一个内核进程持有。当一个内核进程需要使用总线，对某个存储单元进行读写访问时，先**请求自旋锁**，以获得对总线的使用权。

被占用：进程一直“旋转”，**循环测试锁的状态**，直到锁可用。

未被占用：该进程得到它继续执行，直到完成读写操作，释放锁。



3. 自旋锁与信号量的主要差别

自旋锁与信号量的主要差别在于：自旋锁可避免调用进程阻塞。

用自旋锁所保护的临界区一般都应比较短，调用进程用“旋转”来取代进程切换。由于进程切换需要花费一定开销，并且会使高速缓存失效，直接影响系统的性能，因此将自旋锁应用于对总线资源的竞争，其效率远高于信号量机制，且在多处理器环境中非常方便。



信号量

对于被保护的共享资源
仅在进程的上下文
访问；

或有共享设备；

或调用进程所保护的
临界区较大



自旋锁

被保护的共享资源
需要中断上下文访
问；

调用进程所保护的
临界区非常小，对
共享资源的访问时
间非常短

4. 自旋锁的类型

使用自旋锁的基本形式为：

```
spin_lock(&lock);
```

```
/*临界区代码; */
```

```
.....
```

```
spin_unlock(&lock);
```



常用的自旋锁有三种形式：

(1) 普通自旋锁：若是锁可用，则将自旋锁变量置为0，否则为1。该类自旋锁的使用不会影响当前处理机的中断状态，一般在临界区的代码在禁止中断或者不能被中断处理程序所执行时使用。

(2) 读写自旋锁：允许多个读者同时以只读的方式访问相同的共享数据结构，但是当一个写者正在更新这个数据结构时，不允许其它读者或写者访问。

(3) 大读者自旋锁：获取读锁时只需要对本地读锁进行加锁，开销很小；获取写锁时则必须锁住所有CPU上的读锁，代价较高。



8.4.3 读—拷贝—修改锁和二进制指数补偿算法

1. 读—拷贝—修改锁(RCU)的引入

不论是第二章中的读写问题，还是前面所介绍的读写自旋锁，都是允许多个进程同时读，但只要有一个写进程在写，便禁止所有读进程去读，使读者进入阻塞状态。如果写的时间非常长，将严重影响到多个读进程的工作。是否能改善这一情况呢？

解决方法？



2. RCU (Read-Copy-Update) 锁

RCU锁用来解决读者—写者问题。

读者：直接访问

写者：制作副本，使用回调机制

（向垃圾收集器注册一个回调函数，适当时机
由垃圾收集器调用）



3. 写回时机

在RCU锁机构中，如何确定将修改后的内容写回的时机？
显然，最好是在所有读者都**已完成自己的读任务后再将修改后的文件写回**。为此，每一个读者完成对共享文件的操作后，都必须向写者提供一个信号，表示它不再使用该数据结构。

写者：**写延迟期**



4. RCU锁的优点

RCU实际上是一种改进的读写自旋锁。它的主要优点表现为如下两方面：

- (1) 读者不会被阻塞。
- (2) 无需为共享文件(数据)设置同步机构。



RCU一般用于读者较多写操作很少的情况。写者开销比较大。

8.4.4 二进制指数补偿算法和待锁CPU等待队列机构

1. 二进制指数补偿算法

基本思想是：为每一个CPU 对所进行测试的TSL（Test and Set Lock）指令设置一个指令延迟执行时间，使该指令的下次执行是在该延迟执行时间设定的时间后进行，其延迟时间是按照一个TSL指令执行周期的二进制指数方式增加。

例如当一个CPU发出TSL指令对锁进行第一次测试，发现锁不空闲时，便推迟第二次测试指令的执行时间，等到 2^1 个指令执行周期后，如果第二次测试仍未成功，则将第三次指令的执行时间推迟到 2^2 个指令执行周期后，.....，如果第 $n-1$ 次测试仍未成功，则将第 n 次的测试推迟到 2^{n-1} 个指令执行周期后，直到一个设定的最大值；当锁释放时，可能首先由延迟时间最小的CPU获得该锁。

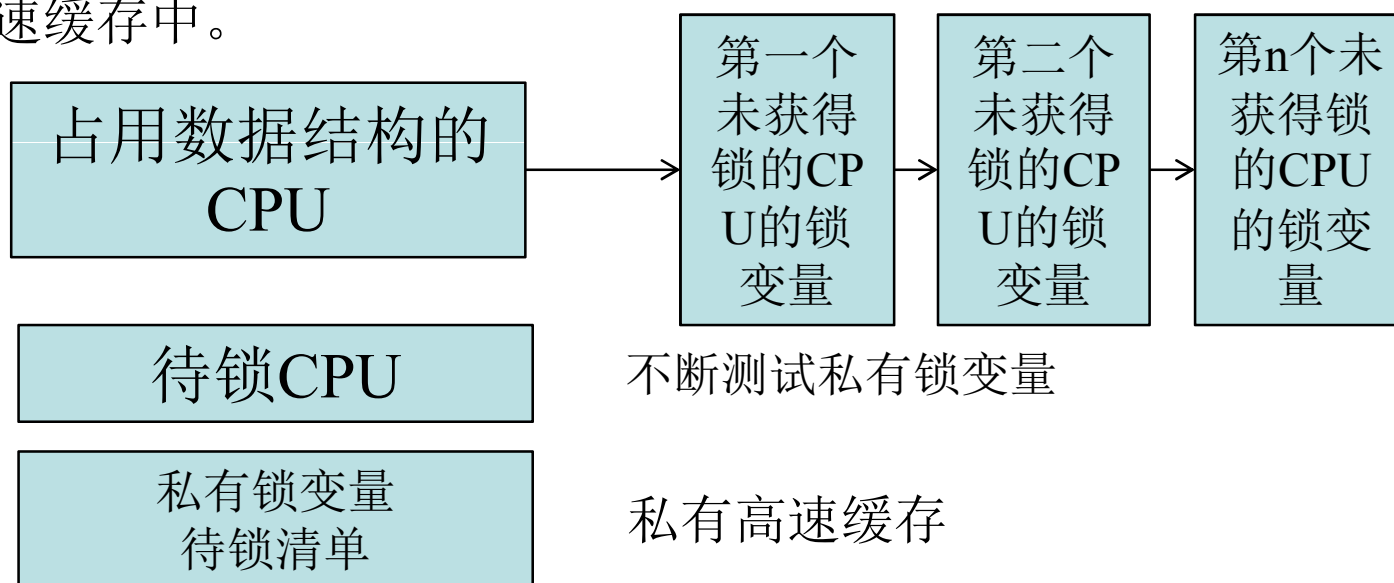


采用二进制指数补偿算法可以明显降低总线上的数据流量。

缺点：锁被释放时，可能由于各CPU的测试指令的延迟时间未到，没有一个CPU会及时的对锁进行测试，**即不能及时地发现锁的空闲，造成浪费。**

2. 待锁CPU等待队列机构

如何及时发现锁空闲，另一种同步机构——**锁等待队列机构**很好地解决了这一问题。这种机构的核心思想是：为每一个CPU配置一个用于测试的**私有锁变量**和一个记录待锁CPU的**待锁清单**，存放在其私有的高速缓存中。





8.4.5 定序机构

在多处理机系统和分布式系统中，有着许多的处理机或计算机系统，每个系统中都有自己的物理时钟。为了能对各系统中的所有特定事件进行排序，以保证各处理机上的进程能协调运行，在系统中应有定序机构。



1. 时间戳定序机构 (Timestamp Ordering Mechanism)

对时间戳定序机构最基本的要求是，在系统中应具有**唯一的、由单一物理时钟驱动的物理时钟体系**，确保各处理机时钟间的严格同步。该定序机构的基本功能是：


- (1) 对所有的特殊事件，如资源请求、通信等，**加印上时间戳**；
- (2) 对每一种特殊事件，只能**使用唯一的时间戳**；
- (3) 根据事件上的时间戳，**定义所有事件的全序**。



2. 事件计数(Event Counts)同步机构

在这种同步机构中，使用了一个称为**定序器 (Sequencers)**的整型量，为所有特定事件进行排序。定序器的初值为0，且为非减少的，对其仅能施加**ticket (S)** 操作。

一个事件发生时，系统便为之分配一个称为编号（或标号）**V**的序号，然后使ticket自动加1，一系列的ticket操作形成了一个非负的，增加的整数序列，然后把打上标号的事件**送至等待服务队列排队**。与此同时系统将所有已服务事件的标号保留，并形成一个称为**事件计数E**的栈。实际上，E是保存已出现的某特定类型事件编号计数的对象 (Object)，其初值为0，当前值是栈顶的标号。事件计数有下面三种操作：



1) await(E, V)

每当进程要进入临界区之前，先执行await操作，如果 $E < V$ ，将执行进程插入到EQ队列，并重新调度；否则进程继续执行。

```
await(E,V){  
    if(E<V) {  
        i=EP;  
        stop();  
        i->status="block";  
        i->sdata=EQ;  
        insert(EQ,i);  
        scheduler();  
    }  
    else continue;  
}
```




2) advance(E)

每当进程退出临界区时，应执行advance(E)操作，使E值增1。如果E Q队列不空，则进一步检查队首进程的V值；若E=V，则唤醒该进程。

```
Advance(eventcount E){  
    E=E+1;  
    if(EQ<>NIL){  
        V=inspect(EQ,1);  
        if(E==V)wakeup(EQ,1);  
    }  
}
```

一个进程执行临界区的操作序列为：

```
await (E,V) ;  
Access the critical resources;  
advance (E);
```



3) read (E)


返回E的当前值，提供给进程参考，以决定是否要转去处理其它事件。如果设计得当，允许await、read和advance这三个操作，在同一事件上并发执行，但对定序器必须互斥使用。



8.4.6 面包房算法


该算法是最早的分布式进程同步算法，是利用事件排序的方法对要求访问临界资源的全部事件进行排序，按照**FCFS**次序对事件进行处理。

银行的排队系统



该算法的基本假定如下：

- （1）系统由 N 个结点组成，每个结点只有一个进程，仅负责控制一种临界资源，并处理那些同时到达的请求。
- （2）每个进程保持一个队列，用来记录本结点最近收到的消息，以及本结点自己产生的消息。
- （3）消息分为请求消息、应答消息和撤销消息三种，每个进程对类中的请求消息根据事件时序排序，队列初始为空。
- （4）进程 P_i 发送的请求消息形如 $\text{request}(T_i, i)$ ，其中 $T_i = C_i$ ，是进程 P_i 发送此消息时对应的逻辑时钟值， i 代表消息内容。



面包房算法描述如下：

(1) 当进程 P_i 请求资源时，它把请求消息 $\text{request}(T_i, i)$ 排在自己的请求队列中，同时也把该消息发送给系统中的其它进程；

(2) 当进程 P_j 接收到外来消息 $\text{request}(T_i, i)$ 后，发送回答消息 $\text{reply}(T_j, j)$ ，并把 $\text{request}(T_i, i)$ 放入自己的请求队列。应当说明，若进程 P_j 在收到 $\text{request}(T_i, i)$ 前已提出过对同一资源的访问请求，那么其时间戳应比 (T_i, i) 小。、

(3) 若满足下述两条件，则允许进程 P_i 访问该资源（即允许进入临界区）：

- P_i 自身请求访问该资源的消息已处于请求队列的最前面；
- P_i 已收到从所有其它进程发来的回答消息，这些回答消息的时间戳均晚于 (T_i, i) 。

(4) 为了释放该资源， P_i 从自己的队列中撤销请求消息，并发送一个打上时间戳的释放消息给其它进程；

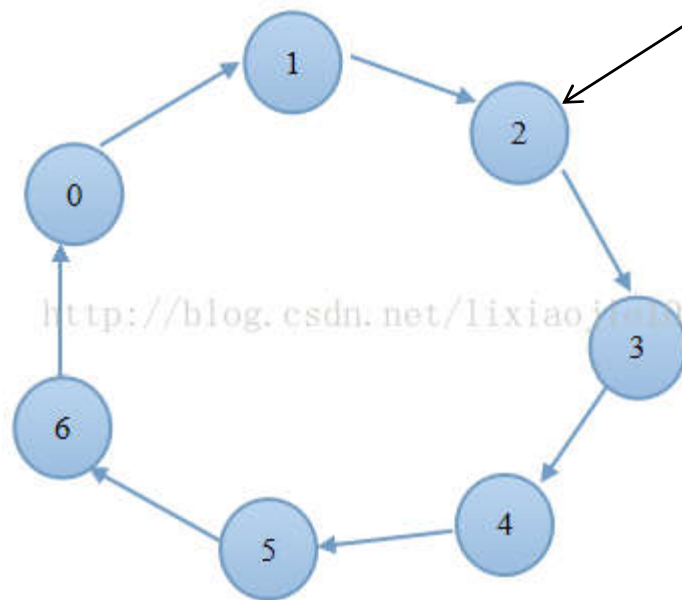
(5) 当进程 P_j 收到 P_i 的 release 消息后，它撤销自己队列中的原 P_i 的 $\text{request}(T_i, i)$ 消息。

8.4.7 令牌环算法

该算法属于分布式同步算法，是将所有进程组成一个逻辑环(Logical Ring)，系统中设置一个象征存取权力的令牌(Token)，它是一种特定格式的报文，在进程所组成的逻辑环中，不断地循环传递，获得令牌的进程，才有权力进入临界区，访问共享资源。

令牌

随机发放





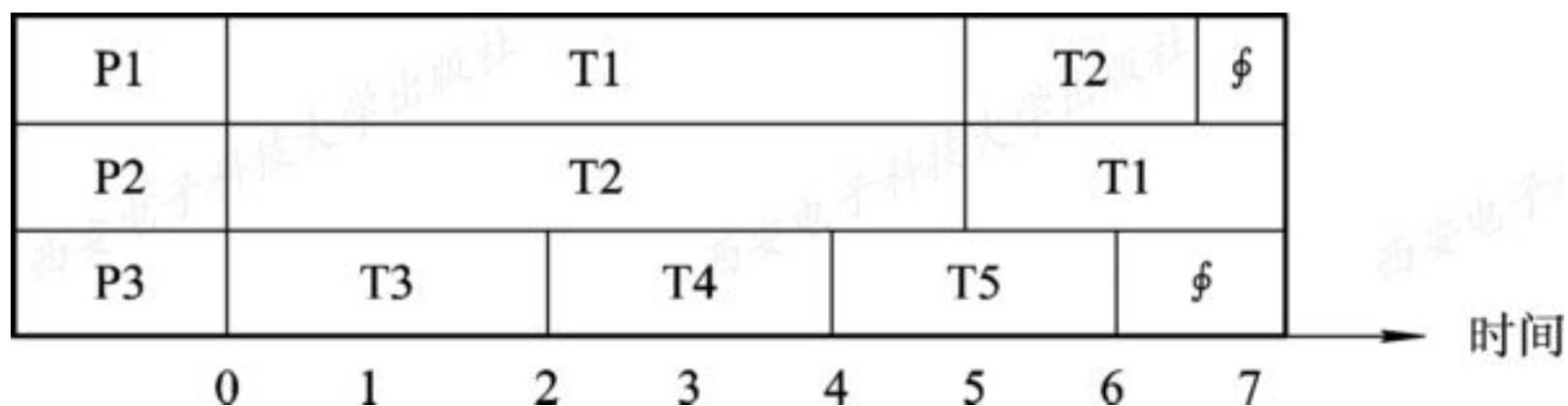
8.5 多处理机系统的进程调度

在多处理机系统中，进程的调度与系统结构有关。例如，在同构型系统中，由于所有的处理机都是相同的，因而可将进程分配到任一处理机上运行；但对于非对称多处理机系统，则只能把进程分配到适合于它运行的处理机上去执行。

8.5.1 评价调度性能的若干因素

1. 任务流时间

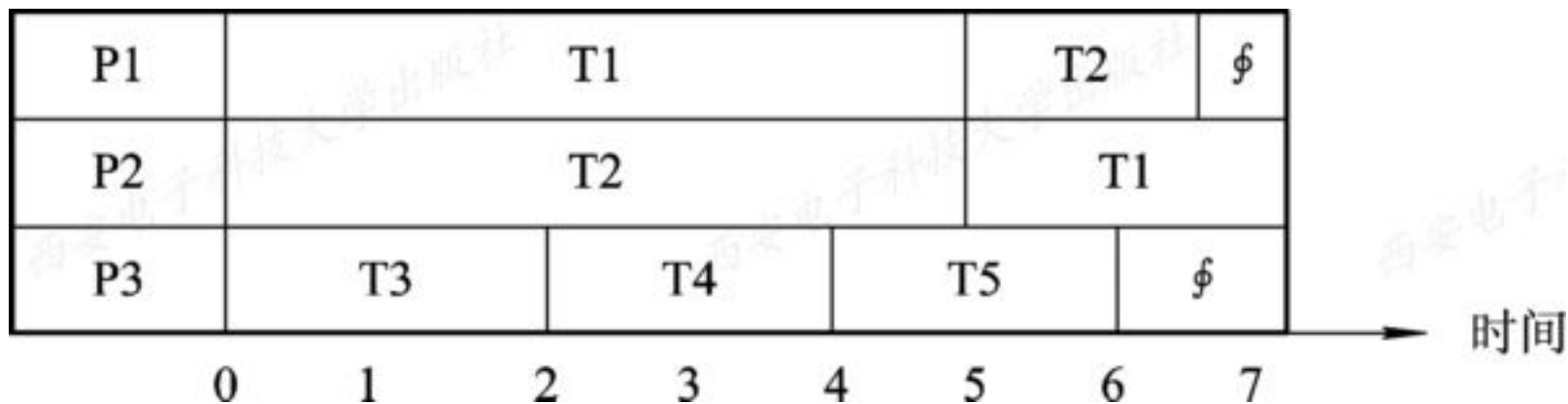
把完成任务所需要的时间定义为任务流时间。



$$T1=5+2=7, \quad T2=5+1.5=6.5$$

2. 调度流时间

在多处理机系统中，任务可以被分配到多个处理机上去运行。一个调度流时间是系统中所有处理机上的任务流时间的总和。



$$\text{调度流时间} = 7 + 6.5 + 2 + 2 + 2 = 19.5$$

8-6 任务流和调度流示意图



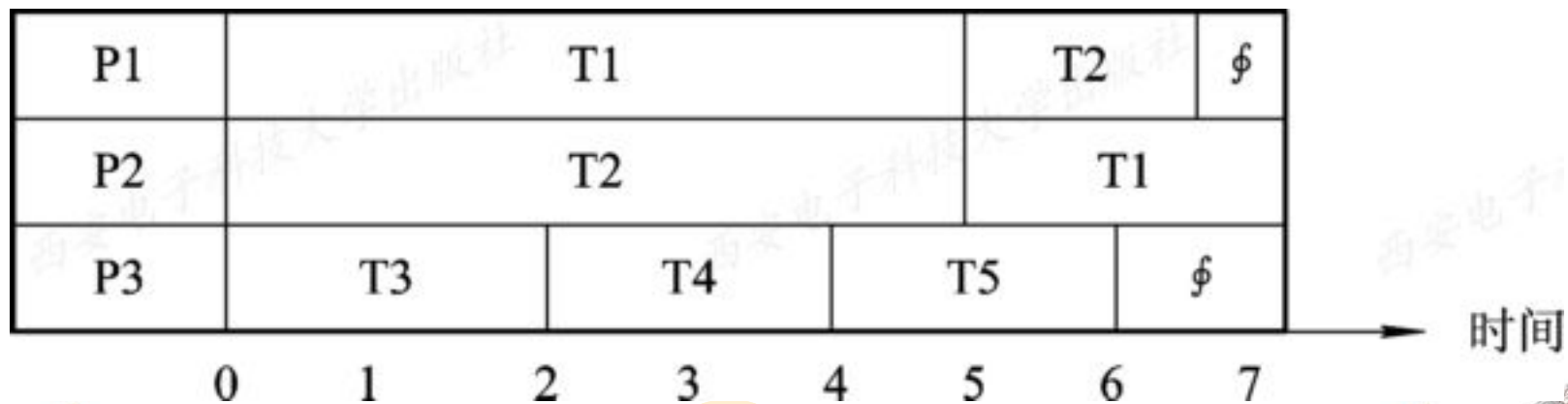
3. 平均流

平均流等于调度流时间除以任务数。平均流时间越小，表示任务占用处理机与存储器等资源的时间越短，系统资源利用率越高，还可以降低任务的机时费用。

更为重要的是，还可使系统有更充裕的时间处理其它任务，有效地提高了系统的吞吐量。因此，最少平均流时间就是系统吞吐率的一个间接度量参数。

4. 处理机利用率

处理机的利用率等于该处理机上任务流之和除以最大有效时间单位。在如图8-6所示的例子中，最大有效时间单位为7.0，三台处理机P1、P2、P3的空闲时间分别为0.5、0.0和1.0，忙时间分别为6.5、7.0、6.0，它们为各处理机上的任务流之和。由此可以得到P1、P2、P3的处理机利用率分别为0.93、1.00和0.86。处理机平均利用率 $= (0.93 + 1.00 + 0.86) \div 3 = 0.93$ 。





5. 加速比

加速比等于各处理机忙时间之和除以并行工作时间，其中，各处理机忙时间之和，相当于单机工作时间，在上例中为19.5个时间单位；并行工作时间，则相当于从第一个任务开始到最后一个任务结束所用的时间，在上例中为7个时间单位。由此得到加速比为19.5个时间单位/7个时间单位。



6. 吞吐率

吞吐率是单位时间(例如每小时)内系统完成任务数。

可以用任务流的最小完成时间来度量系统的吞吐率。



8.5.2 进程分配方式

1. 对称多处理机系统中的进程分配方式

在SMP系统中，所有的处理机都是相同的，因而可把所有的处理机作为一个处理机池(Processor pool)，由调度程序或基于处理器的请求，将任何一个进程分配给池中的任何一个处理机去处理。对于这种进程分配，可采用以下两种方式之一。



1) 静态分配(Static Assigenment)方式

指一个进程从开始执行直至完成，都被固定地分配到一个处理器上执行。

2) 动态分配(Dynamic Assgement)方式

为了防止系统中的多个处理器忙闲不均，可以在系统中仅设置一个公共的就绪队列，系统中的所有就绪进程都被放在该队列中。分配进程时，可将进程分配到任何一个处理器上。



2. 非对称MPS中的进程分配方式


对非对称多处理机系统中大多采用主/从式进程（线程）分配的方式。在主机中保持有一个就绪队列，只要就绪队列不空，主机便从其队首摘取一个进程（线程）分配给请求的从机。当有多个从机同时发出请求，还需考虑处理机类型与计算任务类型，例如科学计算最好在浮点处理器上运行。

主/从式进程（线程）分配方式的主要优点：

因为所有进程的分配由一台主机处理，使进程间的同步问题得以简化，因此，系统处理较简单。

缺点：

但由一台主机控制一切，也潜在着不可靠性，即主机一旦出现故障，将导致整个系统的瘫痪，而且也因为主机太忙，来不及处理而形成系统瓶颈。



8.5.3 进程(线程)调度方式

1. 自调度(Self-Scheduling)方式

1) 自调度机制


自调度方式是**最简单**的一种调度方式。它是直接由单处理机环境下的调度方式演变而来的。在系统中设置有一个**公共的进程或线程就绪队列**，所有的处理器在空闲时，都可自己到该队列中取得一进程(或线程)来运行。



2) 自调度方式的优点

自调度方式的主要优点表现为：

- (1) 容易将单处理机环境下的调度机制移植到多处理机系统中，故它仍然是当前多处理机系统中较常用的调度方式。
- (2) 只要系统中有任务，就不会出现处理机空闲的情况，也不会发生处理机忙闲不均的现象，有利于提高处理机的利用率。



3) 自调度方式的缺点

(1)这种调度必须基于专门的**互斥机制**，保证多处理机不会同时访问系统中唯一的就绪队列。

(2)当处理机**个数较多(如十几个或上百个)**时，对就绪队列的访问可能成为系统的瓶颈；

(3)线程的**两次相继调度**可能被不同处理机选择，使得局部缓冲信息失效；

(4)**不能保证同一进程中的多个线程被同时调度**，可能加长同一应用程序合作进程相互等待的时间。

2. 成组调度 (Gang Scheduling) 方式

基于进程的组调度:将一组相关的进程同时分派到多台处理机上运行, 以减少进程之间相互等待而引起的进程切换, 从而降低系统开销。

线程级别的组调度:将同一进程中的多个线程同时分派到多个处理机上运行, 以减少因相关线程之间的相互等待而引起的切换, 提高线程推进速度, 从而提高系统处理效率。

组调度(gang-scheduling)的思想实际上早于线程。

组调度每次调度都可以解决一组线程的处理器分配问题, 因而可以显著地减少调度频率, 从而也减少了调度开销。

可见, 成组调度的性能优于自调度, 目前已获得广泛的认可, 并被应用到许多种多处理机OS中。

1) 面向所有应用程序平均分配处理器时间

假定系统中有N个处理机和M个应用程序，每个应用程序中至多含有N个线程，则每个应用程序至多可有 $1/M$ 的时间去占有N个处理机。

	应用程序A	应用程序B
处理器1	线程1	线程1
处理器2	线程2	空闲
处理器3	线程3	空闲
处理器4	线程4	空闲
	1/2	1/2

(a) 浪费 37.5%

	应用程序A	应用程序B
处理器1	线程1	线程1
处理器2	线程2	空闲
处理器3	线程3	空闲
处理器4	线程4	空闲
	4/5	1/5

(b) 浪费 15%

图8-7 两种分配处理器时间的方法





2) 面向所有线程平均分配处理机时间

由于应用程序A中有4个线程，应用程序B中只有1个线程，因此，应为应用程序A分配 $\frac{4}{5}$ 的时间，只为应用程序B分配 $\frac{1}{5}$ 的时间，如图8-7(b)所示。此时，将只有15%的处理机时间被浪费。可见，按线程平均分配处理机时间的方法更有效。



3. 专用处理机分配(Dedicated Processor Assignment) 方式

1989年Tucker提出了专用处理机分配方式。该方式是指在一个应用程序的执行期间，专门为该应用程序分配一组处理机，每一个线程一个处理机。这组处理机仅供该应用程序专用，直至该应用程序完成。很明显，这会造成处理机的严重浪费。但这种调度方式可以用于并发程度相当高的多处理机系统。



Tucker在一个具有16个处理机的系统中，运行两个应用程序：一个是矩阵相乘程序，另一个是进行快速傅立叶变换（FFT）。每个应用程序所含有的线程数是可以改变的，从1到24个。

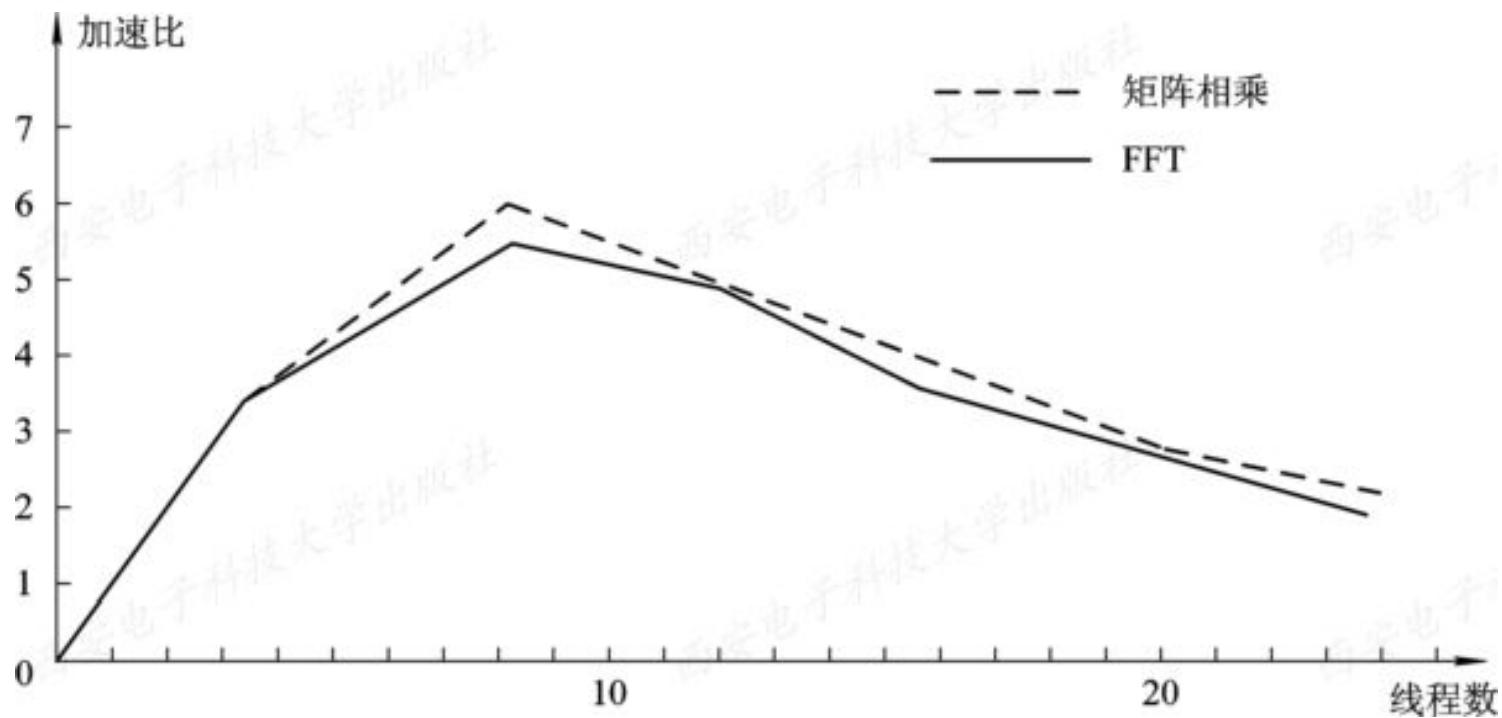



图8-8 线程数对加速比的影响

Tucker建议：在同时加工的应用程序中，其线程数的总和，不应超过系统中处理机的数目。





4. 动态调度

该调度方式允许进程在执行期间动态地改变其线程的数目。这样，操作系统和应用程序能够共同地进行调度决策。

操作系统：负责把处理机分配给作业，

每个作业：负责将分配到的处理机再分配给自己的某一部分可运行任务。



在这种方法中，操作系统的调度责任主要**限于处理机的分配**，并遵循以下的原则：

(1) **空闲则分配**。当一个或多个作业对处理机提出请求时，如果系统中存在空闲的处理机，就将它分配给这个作业，满足作业的请求。

(2) **新作业绝对优先**。

(3) **保持等待**。如果一个作业对处理机的请求，系统的任何分配都不能满足，作业便保持未完成状态，直到有处理机空闲，可分配。

(4) **释放即分配**。当作业释放了一个或多个处理机时，将为这个或这些处理机扫描处理机请求队列，首先为新作业分配处理机，其次按先来先服务原则，将剩余处理机进行分配。

动态调度方式优于成组调度和专用处理机调度方式，但其开销比较大。需要精心设计具体的调度算法。



8.5.4 死锁

在多处理机系统中，产生死锁的原因以及对死锁的防止、避免和解除等基本方法与单处理机相似，但难度和复杂度增加很多。尤其是在NUMA分布式环境下，进程和资源在配置和管理上呈现了分布性，竞争资源的各个进程可能来自不同结点。每个资源结点，通常只记录本结点的资源使用情况，因此，来自不同结点的进程在竞争共享资源时，对于死锁的检测显得十分困难。



1. 死锁的类型

资源死锁：前者是因为竞争系统中可重复使用的资源(如打印机、磁带机、存储器等)时，由于进程的**推进顺序不当**引起的。

通信死锁：分布式系统中，由于处于不同结点中的进程，因**发送和接收报文**而竞争缓冲区引起的，如果出现了既不能发送又不能接收的僵持状态，即发生了通信死锁。

2. 死锁的检测和解除

1) 集中式检测

处理机进
程资源图

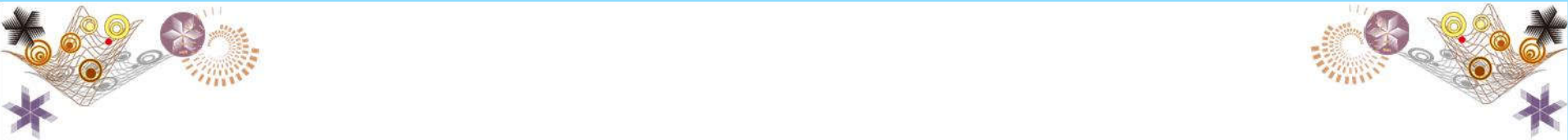
每台处理机

系统进程
资源图

中心处理机

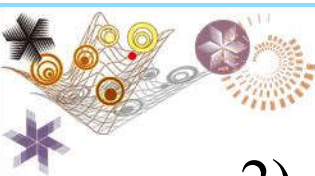
检测进程

检测到环路时，就选择终止环路中的一个进程，解决死锁



为了及时获得最新的进程和资源状况，检测进程对各个结点更新信息的获得，可以通过三种方式：

- ①当资源图加入或删除一条弧时，相应的变动消息就发送给检测进程；
- ②每个进程将新添加或删除的弧的信息周期性地发送给检测进程；
- ③检测进程主动去请求更新信息。




2) 分布式检测

该方式在每个结点中都设置一个死锁检测进程，在每个消息上附加逻辑时钟，并依次对请求和释放资源的消息进行排队，

在一个进程对某资源操作前，必须先向所有其它进程发送请求信息，在获得这些进程的响应信息后，才把请求资源的消息发给该资源的管理进程。每个进程都要将资源的已分配情况通知所有进程。

由此可见，对于分布式环境下的死锁检测，需要的通信开销较大，在实际应用中，往往采取的是死锁预防方式。

（死亡等待法，受伤等待法）




1.死亡等待法

开始是T1占用R，T2也试图占用R。假定T1的时间戳是TS1，T2的是TS2。

如果事务T2在事务T1之前开始(按照时间戳值)，那么T2等待T1结束。否则，分布式操作系统就会杀死T2，并在一段时间后重新启动它(具有相同的时间戳TS2)，同时假设T1目前已经释放了资源R。

事务发生越早，优先级越高。同样可以看到，被杀死的进程重新启动时，它的时间戳值是最初的时间戳值。那样该进程就可以保持它以前的优先级(在重新启动该进程时，它的优先级将比大部分其他事务的优先级高)。




2. 受伤等待法

相比于前面的方法，该方法采用了不同的技术。这里，开始也是先进行类似的比较。比较T1和T2的时间戳(也就是TS1和TS2)。如果TS2小于TS1(意味着T2在T1之前先运行)，现在杀死T1。如果T1开始得比较早，那么就停止T2。



8.7 分布式文件系统



分布式文件系统是配置在分布式系统上的，因此，本节在介绍分布式文件系统这个主题前，先对所涉及的分布式系统的知识和概念做个扼要的介绍。



8.7.1 分布式系统



1. 分布式系统的特征

分布式系统(distributed system), 是基于软件实现的一种多处理机系统, 是多个处理机通过通信线路互连而构成的松散耦合系统, 系统的处理和控制在各个处理机上。换言之, 是利用软件系统方式构建在计算机网络之上的一种多处理机系统。



与前面所述的多种多处理机系统(包括多处理机和多计算机等)相比, 分布式系统的不同在于:

- ① 分布式系统中的每个节点都是一台独立的计算机, 并配置有完整的外部设备;
- ② 分布式系统中节点的耦合程度更为分散, 地理分布区域更加广阔;
- ③ 分布式系统中的每个节点可以运行不同的操作系统, 每个节点拥有自己的文件系统, 除了本节点的管理外, 还有其它多个机构对其实施管理。



对分布式系统有很多不同的定义，

“一个分布式系统是一些独立的计算机集合，但是对这个系统的用户来说，系统就像一台计算机一样”，

“分布式系统是能为用户自动管理资源的网络操作系统，由它调用完成用户任务所需要的资源，而整个网络像一个大的计算机系统一样对用户是透明的。”等等。



分布式系统应具有以下几个主要特征：

(1) 分布性。地理位置的分布，功能的分布，资源的分布和控制的分布（自治性）

(2) 透明性。每台计算机用户都可以使用系统中的所有资源

(3) 同一性。系统中的若干台计算机可以相互协作完成同一个任务，一个程序可以并行运行。

(4) 全局性。系统具备全局性的通信机制。



2. 分布式系统的优点



分布式系统与集中式系统相比具有以下一些优点：

- (1) 计算能力强。
- (2) 易于实现共享。
- (3) 方便通信。
- (4) 可靠性高。
- (5) 可扩充性好。



3. 分布式操作系统

分布式操作系统是配置在分布式系统上的公用操作系统，以全局的方式对分布式系统中的所有资源进行统一管理，可以直接对系统中地理位置分散的各种物理和逻辑资源进行动态的分配和调度，有效地协调和控制各个任务的并行执行，协调和保持系统内的各个计算机间的信息传输及协作运行，并向用户提供一个统一的、方便的、透明的使用系统的界面和标准接口。一个典型的例子是万维网(World Wide Web)，在万维网中，所有的操作只通过一种界面——Web页面。



分布式操作系统除了涵盖单机操作系统的主要功能外，还应该包括：

- (1) 通信管理功能。（通信原语的方式）
- (2) 资源管理功能。
- (3) 进程管理功能。（进程或计算迁移，分布式同步和互斥，应对死锁的措施）

10.7.2 分布式文件的实现方式和基本要求

1. DFS的实现方式

DFS有多种实现方式，一般分为以下两类：

1) 共享文件系统方式(shared file system approach)

该方式也称**专用服务器方式**。类似于本地文件系统使用树形目录结构，管理本地计算机存储设备上的文件方式。共享文件系统方式也**采用一个逻辑树的结构，对整个系统中的文件系统进行管理。**

系统采用客户机服务器模式，设置了若干个文件服务器，数据分布地存储在各个文件服务器上。用户可以忽略文件的实际物理位置，像访问本地文件一样访问分布在多个节点上的文件。

该方式实现比较简单，通用性比较好。



2) 共享磁盘方式(shared disk approach)

该方式也称为无服务器方式。在这种方式中，系统中没有专门的文件服务器，而是配置了一个共享磁盘(一般为高速磁盘，如IBM SSA)，并将其与主机、客户机都连接在内部的高速网络(如光通道)上，主机和客户机都将共享磁盘作为它们的存储设备，直接以盘块方式读写磁盘上的文件，实现共享。

该方式对设备要求比较高，被用来构造高端或专用的存储设备。NAS， SAN

NAS (Network Attached Storage) 网络附属存储，将存储设备与服务器彻底分离) **NAS (Network Attached Storage: 网络附属存储)** 连接在网络上，具备资料存储功能的装置，因此也称为“网络存储器”。它是一种专用数据存储服务器。它以数据为中心，将存储设备与服务器彻底分离，集中管理数据，从而释放带宽、提高性能、降低总拥有成本、保护投资。其成本远远低于使用服务器存储，而效率却远远高于后者。目前国际著名的NAS企业有Netapp、EMC、OUO等。



存储区域网络（Storage Area Network，简称SAN）采用网状通道（Fibre Channel，简称FC，区别与Fiber Channel光纤通道）技术，通过FC交换机连接存储阵列和服务器主机，建立专用于数据存储的区域网络。



图2 典型的SAN环境



2. 基本要求

相对于LFS，DFS除了大容量的要求外，还有很多基本要求：

(1) 透明性。

位置透明，移动透明，性能透明，扩展透明。

(2) 高性能和高可靠性。

(3) 容错性。采用冗余技术掩盖故障。（信息冗余，时间冗余，物理冗余。

(4) 安全性。（身份验证、访问控制、安全通道）

(5) 一致性。本地缓存的文件副本与文件服务器上的副本一致。




10.7.3 命名及共享语义

1. 命名

如前面章节所述，对于OS管理的存储资源，文件系统通过抽象，屏蔽了对物理设备的操作以及资源管理的细节，并向用户提供统一的、对象化的访问接口。



在DFS中，从逻辑对象到物理对象之间，还存在着磁盘所在的服务器地址问题。作为DFS的抽象功能，不仅要将在磁盘的**具体盘块、磁道和扇区等信息隐藏起来**，将其与文件名映射，完成一次文件的抽象，还需要**隐藏文件所在服务器的地址和存储方式等细节**，将其也与文件名映射，从而形成一个多级映射，为用户提供的是一个文件的抽象。**命名就是在数据的逻辑对象和物理对象之间建立的映射。**





DFS中有三种命名方案：

- （1）结合主机名和本地名对文件命名，保证在整个系统范围的唯一性。（简单，但不符合透明性）
- （2）将若干台服务器中的远程目录，加载到客户机的本地目录中。该方案管理复杂度很高，结构混乱，且安全程度低。
- （3）全局统一命名，即系统采用统一的全局命名结构，每个文件和目录使用唯一的命名。由于不同系统有特殊文件，该方案实现难度较大。



2. 共享语义

对于DFS，需要保证多个客户机并发访问时的数据一致性，因此，当多个客户共享同一个文件时，必须对客户机和服务器之间的交互协议精确处理，即精确定义读和写的语义。





客户机A

客户机B




正确语义：只有当客户机**A**对文件修改完毕，执行关闭操作后，才将修改后的副本上传到服务器，并通知客户机**B**。这样，客户机**B**进行后续的读操作，确保了**B**读到正确的文件。





实现方法主要有四种：

- （1）**UNIX** 语义，文件上的每个操作队所有进程都是瞬间可见；
- （2）会话语义，在文件关闭之前，所有改动对其进程都是不可见的；
- （3）不允许更新文件，不能进行更改，只能简单的进行共享和复制；
- （4）事务处理，所有改动以原子操作的方式发生。



3. 租赁协议

租赁协议是一个比较具有代表性的一致性访问协议。当客户机向服务器发出一个读请求时，不仅收到所请求的数据，还会收到一个租赁凭据。该凭据附带一个有效期，保证服务器在该有效期内不会对客户机收到的数据进行更新。



如果客户机数据在本地高速缓存上保留了副本，则在有效期内，客户机对数据的访问只需在本地缓存中进行，不必重新请求服务器，数据也不用重新传送。只有当租赁的有效期过期时，客户机才必须与服务器进行交互，确认本地缓存中的数据是否与服务器的一致，判断服务器上的数据是否进行了更新。只有确认发生了更新，客户机才必须重新向服务器请求，以获得新的数据。

作为服务器，当有客户机对数据发出更新请求时，并不是立即进行更新，而是向所有持有对该文件租赁的客户机发出更新确认，然后才会实施更新。所有客户机在收到服务器发送的更新确认时，即将其持有的租赁凭据标记为无效。此后，客户机需要再次访问该数据时，必须重新向服务器请求，在获得新数据的同时，获取一张新的租赁凭证。



8.7.4 远程文件访问和缓存



在DFS中，客户使用远程服务机制访问文件，访问请求被送到服务器，服务器执行访问把结果送回给客户。在DFS中引入了缓存机制，如果客户请求的数据不在本地，则从服务器处取来数据的拷贝，送给客户机。随后对文件的访问可以在本地副本中进行。



1. 缓存和远程服务的比较



(1) 使用缓存时，大量的远程访问可转为对本地的缓存访问，因此而获得的服务速度与本地访问的一样快。

(2) 使用缓存时，服务器的负载和网络通信量都减少了，扩充能力加强了。而在使用远程服务方法时，每次远程访问都是跨过网络处理的，明显增加了网络通信量和服务器负载，引起性能下降。



(3) 缓存时，就网络总开销而言，与远程服务针对个别请求一系列应答的传输开销相比，缓存采用以整个文件或文件的若干个页面这样的大批数据传输方式时，开销还是要低很多。

(4) 缓存的主要缺点是一致性问题。在针对不经常写入的访问模式中，缓存方法是优越的；但在频繁写的情况下，用于解决一致性问题的机制反而导致在性能、网络通信量和服务器负载等方面的大量开销。



(5) 在用缓存作为远程访问方法的系统中，仿真集中式系统的共享语义是很困难的。使用远程服务时，服务器将所有访问串行化，因此能够实现任何集中的共享语义。

(6) 机器间的接口不同，远程服务方式仅仅是本地文件系统接口在网络上的扩展，机器间的接口就是本地客户机和文件系统之间的接口。而缓存方式中，数据是在服务器和客户机之间整体传输，机器间的接口与上级的用户接口是不同的。




2. 缓存的粒度和位置

1) 缓存的粒度

在DFS中，缓存的**数据粒度**(即数据单元)可以是文件的若干块，也可以是若干个文件，乃至整个文件系统。

缓存粒度大，增加访问速度，减少通信流量。
增加一次性传输的开销和保持数据一致性上的开销。

缓存粒度小，降低缓存的命中率，增加了通信开销，加大了服务器负载，降低了客户机的访问速度。




2) 缓存的位置

在一个各自有主存和磁盘的客户-服务器系统中，有四个地方可以用来存储文件或存储部分文件：**服务器磁盘、服务器主存、客户机磁盘或者客户机主存。**

存储所有文件最直接的位置是在服务器磁盘上，使用磁盘缓存最明显的优点就是**可靠性**，不会因为系统的故障而丢失。

也可以利用主存做缓存器。可支持无盘工作站，速度快，方便构造单缓存机制。



3. 缓存的更新

对于缓存中更新的数据块，选择什么时间和方式将其写回服务器，对系统的性能和可靠性具有关键性的影响。

有几种写回策略：

- (1) 直接写
- (2) 延迟写（可能语义不清）
- (3) 驱逐时写（当被修改的数据块将被从缓存中换出时写到服务器上）
- (4) 周期性写
- (5) 关闭时写。（与会话语义相对应）





4. 数据一致性

对于本地缓存的数据副本与服务器中的主副本，客户机需要进行有效性检查，判断它们是否一致后才能使用。

如果不一致，则表明本地缓存的数据已经过时，那么这些数据就不能为客户机提供数据访问服务，需要进行更新。

检查有两个基本方法：客户机发起和服务器发起。



客户机发起：客户机与服务器联系，检查本地数据与服务
器上的主副本是否一致。**频度范围：**每次访问前都进行一
次；只对一个文件的第一次访问时进行，也可以按照固定
的时间间隔周期性进行、检查频率的高低直接影响网络和
服务器的负载。

服务器发起：服务器为每个客户机记录其缓存的文件，当
服务器检测出可能不一致时，必须做出处理，如检测到一个
文件在多个客户机竞争状态被打开时，服务器使该缓存
失效。



8.7.5 容错

1. 无状态服务和有状态服务

当客户机对远程文件进行访问时，有关被访问的文件、目录和客户机的信息等，在服务器端是否需要进行跟踪、检查和保存等处理，存在着两种策略：

(1) 有状态服务(stateful file service)：服务器缓存客户机的有关信息。

(2) 无状态服务(stateless file service)：服务器没有缓存客户机的有关信息。



2. 容错性

上述两种策略对系统的容错性能有不同的影响，当服务器在一次数据服务过程中发生崩溃时，两者之间的差别就尤其明显：

前者将丢失所有易失状态，需要有一种机制将其恢复到崩溃前的状态：

后者在系统崩溃后可方便的重新向客户机提供数据服务。

然而，也正是因为后者没有保留客户机的任何信息，所以需要更长的请求消息和更久的处理过程。



有关DFS的容错性环境，定义了三种文件属性：

(1) 可恢复性

当对某个文件的操作失败，或由客户中断此操作时，如果文件能转换到原来的一致性状态，则此文件是可恢复的。（可通过更新原子操作来保证）



(2) 坚定性

如果当某个存储器崩溃或存储介质损坏时某个文件能保证完好，则说明此文件是坚定的。（可通过设备的冗余性来保证）

(3) 可用性



如果无论何时一旦需要就可访问，甚至在某个机器和存储器崩溃，或者在发生通信失效的情况下，某个文件仍然可以被访问，则称这种文件是有用的。







3. 可用性与文件复制

文件复制是保证可用性的一个冗余措施。这里的文件复制不是SFT三级容错技术中的磁盘镜像(同一台机器不同介质上的文件复制), 而是在DFS系统不同节点的主机磁盘上的复制。





通过对每个文件在多个服务器上的独立备份，增加系统的可靠性，这样当一个文件服务器出现问题时，仍允许通过其他文件服务器进行文件访问。

另外，同一文件存在于多个文件服务器上，当多个客户并行发出文件访问请求时，还可以把这些请求进行分流，分发到这些服务器上，平衡了每个服务器的负载，避免了运行性能上的瓶颈。



文件复制机制设计过程中，必须对用户透明。一个文件多个副本的存在，在进行文件访问服务过程中，由文件名映射到一个特定的副本是命名机制的任务，对高层应该是不可见的。

在底层，不同的副本必须用不同的标识符区分。但是对于用户而言，对文件复制的控制又应该是可控的，即复制控制的一些功能应该设计在高层。



文件复制机制设计中另外一个重要问题就是文件副本的更新。从用户的观点而言，一个文件的所有副本对应的都是同一个逻辑实体，所以，对于其中任何一个副本的更新，必须引发所有副本的更新。

可见，使用文件复制增加可用性的代价，就是要使用一个更为复杂的更新协议，以及系统为保证一致性而需付出的大量开销。



习 题 ►►►

1. 为什么说依靠提高**CPU**时钟频率提高计算机运算速度的方法已接近了极限？
2. 试说明引入多处理机系统的原因有哪些。
3. 什么是紧密耦合**MPS**和松弛耦合**MPS**？
4. 何谓**UMA**多处理机结构？它又可进一步分为哪几种结构？
5. 试说明基于单总线的**SMP**结构和多层总线的**SMP**结构。



6. 试说明使用单级交叉开关的系统结构和使用多级交换网络的系统结构。

7. 什么是**NUMA**多处理机系统结构？它有何特点？

8. 为什么要为每个**CPU**配置高速缓冲区？**CC-NUMA**和**NC-NUMA**所代表的是什么？

9. 试说明多处理机操作系统的特征。

10. 试比较在单处理机**OS**和多处理机**OS**中的进程管理

。

11. 试比较在单处理机**OS**和多处理机**OS**中的内存管理

。

12. 何谓中心同步实体、集中式同步机构和非集中式同步机构？



13. 集中式同步算法具有哪些特征和缺点？

14. 一个完全分布式同步算法应具有哪些特征？

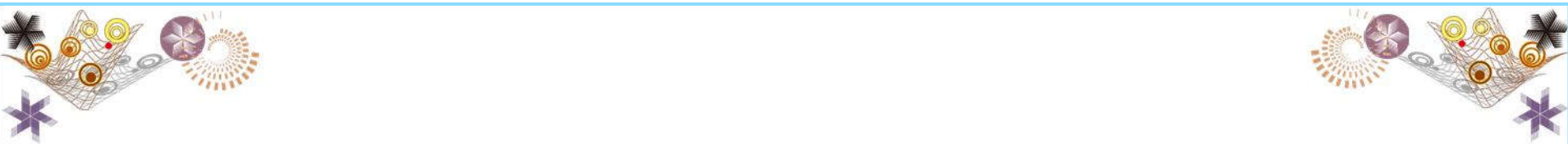
15. 如何利用自旋锁来实现对总线的互斥访问？它与信号量的主要差别是什么？

16. 为什么要引入读—拷贝—修改锁(RCU)？它对读者和写者分别有何影响？

17. 何谓二进制指数补偿算法？它所存在的主要问题是什

18. 时间戳戳定序机构和事件计数的作用是什么？

19. 什么是任务流时间和调度流时间？请举例说明之。



20. 试比较多处理机系统中静态分配方式和动态分配方式。

21. 何谓自调度方式？该方式有何优缺点？

22. 何谓成组调度方式？按进程平均分配处理器和按线程平均分配处理器时间的方法，哪个更有效？

23. 试说明采用专用处理器分配方式的理由。

24. 在动态调度方式中，调度的主要责任是什么？在调度时应遵循哪些原则？

THE END