Chapter 3 处理机调度与死锁

Created	@November 20, 2021 1:51 PM
Tags	

处理机调度与死锁

基本概念

作业与进程

1．作业是用户向计算机提交任务的**任务实体**；进程是完成用户任务的**执行实体**，是资源分配的基本单位

2．作业建立完成之后，放在外存等待运行；进程一旦创建，总有部分存于内存

3．一个作业至少由一个进程组成，反之不然。

4．作业更多用于批处理系统，进程用于所有的多道程序系统

批处理作业调度

- 作业调度

> 使作业进入主存

- 进程调度

> 使作业进程占用处理器

调度的三个层次

1．高级调度（作业调度）

> 按一定的规则在外存上后备队列中挑选一个作业调入内存并创建进程，分配必要的资源并将他们放入就绪队列。每个作业只调入一次，调出一次。

在批处理系统中因为作业进入系统后先驻留外存所以需要作业调度。而分时系统命令和数据直接送入内存，不需要作业调度，实时系统也不需要作业调度。

2．低级调度（进程调度）

> 决定就绪队列中哪个进程可以获得处理机，再由分派程序把处理机分配给该进程。是最基本的一种调度，进程调度的频率很高，三种类型OS都必须要有进程调度

任务

1．保存处理机的现场信息

2．按某种算法选取进程

3．把处理机分配给进程

基本机制

1．排队器

2．分派器

3．上下文切换器

两种方式

1．非抢占式

> 即便有更紧急的进程到达，仍然等待当前进程终止或主动进入阻塞态

2．抢占式

> 时间片原则
>
> 短作业优先原则
>
> 优先权原则

3．中级调度(内存调度)

> 内存不够时可以将某些进程的数据调出外存，等内存空闲再调入，调入外存的进程状态为挂起状态，被挂起的进程PCB被组织成挂起队列。重新调入内存后变成**就绪状态**，挂在就绪队列上

三种调度的联系

1．进程调度频率最高，不可以太复杂

2．作业调度发生在作业运行完毕时，重新调度一个作业周期长

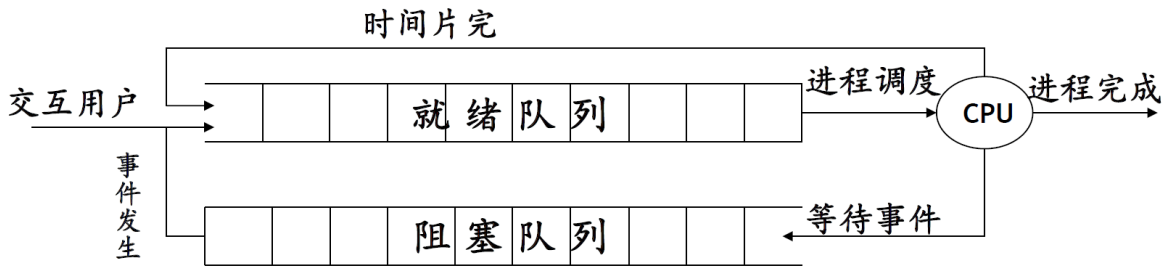
3．中级调度的频率介于两者之间

	要做什么	调度发生在..	发生频率	对进程状态的影响
高级调度（作业调度）	按照某种规则，从后备队列中选择合适的作业将其调入内存，并为其创建进程	外存→内存	最低	无→创建态→就绪态
中级调度（内存调度）	按照某种规则，从挂起队列中选择合适的进程将其数据调回内存	外存→内存	中等	禁止就绪→活动就绪 （禁止阻塞→活动阻塞）
低级调度（进程调度）	按照某种规则，从就绪队列中选择一个进程为其分配处理机	内存→CPU	最高	就绪态→运行态

调度队列的模型

1．只有进程调度的调度队列模型

采用FIFO队列形式，按照时间片轮转方式运行

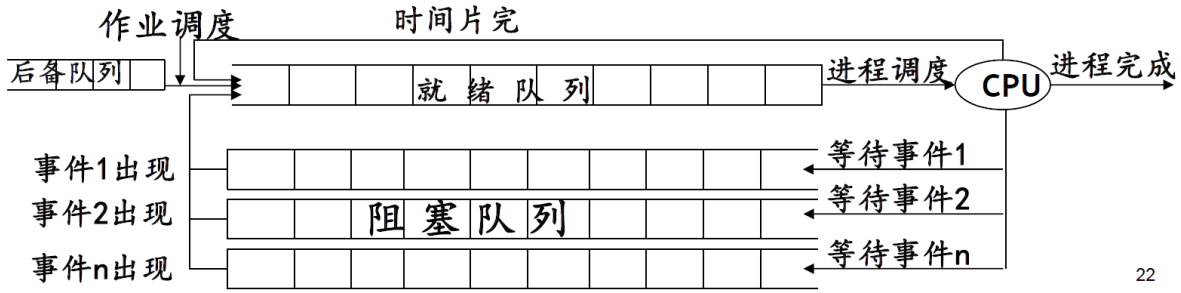


2．具有高级调度和低级调度的队列模型

不仅需要进程调度还需要作业调度，采用某种进程调度算法

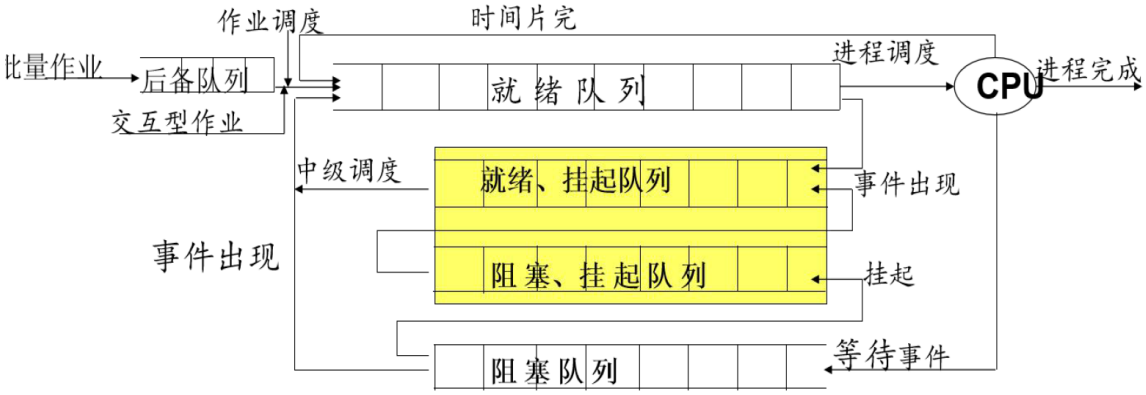
1．就绪队列形式（常用最高优先权有限算法）

2．设置多个阻塞队列



3． 三级调度队列模型

引入中间调度之后，就绪就会分为内存就绪和外存就绪，阻塞分为内存阻塞和外存阻塞



进程调度的时机

- 需要进程调度与切换的情况
  - 主动放弃处理机
    - 进程正常终止
    - 运行过程发生异常而终止
    - 进程主动请求阻塞
  - 被动放弃处理机
    - 分给进程的时间片用完

    - 有更紧急的需要处理（如：IO中断）
    - 有更高优先级的进程进入就绪队列
- 不能进程调度与切换的情况
    - 在处理中断的过程中
    - 进程在操作系统的内核程序临界区中
    - 在原语中

评价算法的指标

1．CPU利用率

$$CPU\text{利用率} = \frac{CPU\text{有效工作时间}}{CPU\text{有效工作时间} + CPU\text{空闲时间}}$$

2．系统吞吐量

$$\text{系统吞吐量} = \frac{\text{完成的总的作业数}}{\text{总共花费的时间}}$$

3．周转时间：作业提交给系统到作业完成为止的时间间隔（越小越好）

> 作业在外存后备队列上等待高级调度的时间，进程在就绪队列上等待低级调度的时间，在CPU上的执行时间，等待IO完成的时间

$$\text{周转时间} = \text{作业完成时间} - \text{作业提交时间}$$

$$\text{平均周转时间} = \frac{\text{各作业周转时间之和}}{\text{作业数}}$$

$$\text{带权周转时间} = \frac{\text{作业周转时间}}{\text{作业实际运行时间}}$$

$$\text{平均带权周转时间} = \frac{\text{各作业带权周转时间之和}}{\text{作业数}}$$

4．等待时间

5．响应时间

6．截至时间

调度算法

> 在OS中调度的实质是一种资源的分配，调度算法是指根据操作系统的资源分配策略所规定的资源分配算法
>
> 在批处理系统中采用**短作业优先**
>
> 分时系统中使用**时间片轮转算法**

算法	优缺点	是否可抢占	是否会饥饿
FCFS先来先服务	有利于长作业，不利于短作业	否	否
SJ(P)F 短作业(进程)优先	有效降低平均等待时间，提高吞吐量；对长作业不利，未考虑作业紧迫程度	否（有抢占式版本）	会
HRRN高响应比优先调度	综合考虑作业进程的等待时间和要求服务的时间	否	不会
时间片的轮转调度算法	公平响应快		

1．FCFS先来先服务（作业，进程）

2．SJ(P)F短作业（进程）优先

3．HRRN高响应比优先（作业，进程）

$$优先级 = \frac{等待时间 + 要求服务时间}{要求服务时间}$$

$$响应比 = \frac{等待时间 + 要求服务时间}{要求服务时间}$$

> 批处理系统中常用1，2，3算法

4．时间片轮转算法（进程调度）

FIFO+时间片轮转

默认新到达的进程优先进入队列

5．优先级调度算法（进程，作业）

常用于批处理系统中作为作业调度算法，也作为操作系统中进程调度算法，还可以用于实时系统

- 抢占式

> 用于批处理

- 非抢占式

> 实时系统

确定优先级的依据

1．进程类型：系统进程高于用户进程

2．进程对资源的需求

3．用户要求

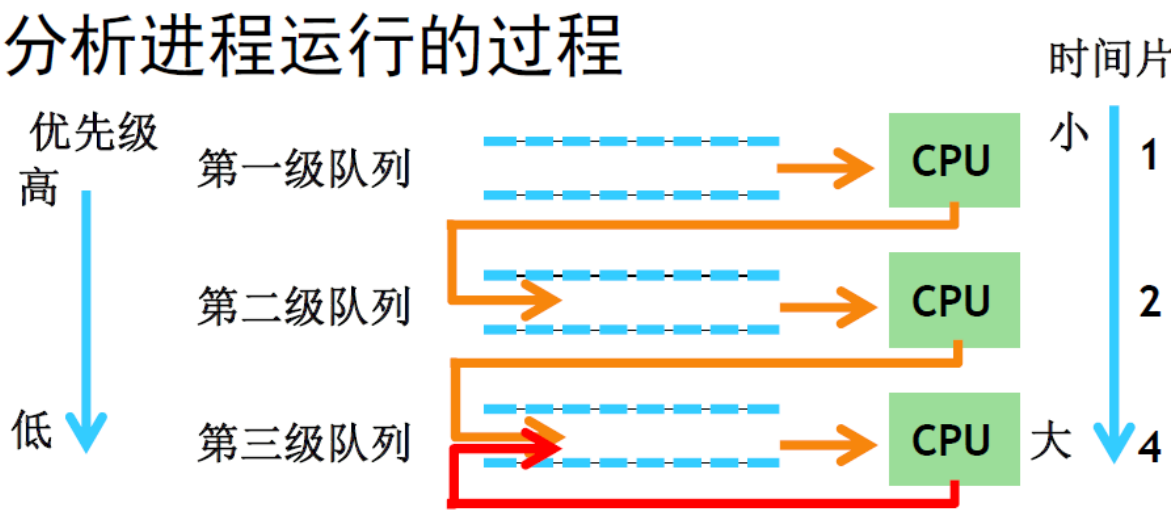
优先级分类

1．静态优先级：进程创建时确定

2．动态优先级：随着等待时间变化

6．多级队列反馈（进程）

设置多个就绪队列，第一级采用FCFS，最后一级使用时间片轮转，被抢占的进程重新放回原队列队尾，时间片结束的进程进入下一级队尾



> 分时和实时系统常用4，5，6算法

实时调度

基本条件

1. 提供必要的信息
    a. 就绪时间
    b. 开始截至时间，完成截至时间
    c. 处理时间
    d. 资源要求
    e. 优先级
2. 系统处理能力强
3. 采用抢占式机制
4. 具有快速切换能力

算法分类

1. 硬实时和软实时
2. 非抢占和抢占
    a. 非抢占
        i. 非抢占式轮转调度算法
        ii. 非抢占式优先调度算法
    b. 抢占
        i. 基于时钟中断的抢占式优先权调度算法
        ii. 立即抢占的优先权调度算法
3. 静态调度和动态调度
4. 集中式调度和分布式调度

常见算法

1. 最早截至时间优先EDF
2. 最低松弛度优先LLF

死锁

定义

> 各进程相互等待对方手里的资源，导致各进程都阻塞，无法向前推进

死锁与饥饿的区别

1. 饥饿是长期得不到想要的资源，导致进程无法推进。
2. **死锁至少是两个进程，且死锁进程一定是阻塞态**
3. **饥饿可以只有一个进程，可能处于阻塞态和就绪态**

产生死锁的原因

1. 竞争资源
    a. 竞争不可抢占资源
    b. 竞争临时资源
2. 进程间推进顺序非法

死锁的必要条件

1. 互斥条件：只有对互斥资源的争抢才会导致死锁
2. 请求和保持条件
3. 不可抢占条件
4. 循环等待条件

> 发生死锁一定有循环等待，循环等待不一定死锁

死锁处理

1. 预防死锁（破坏四个条件中的2，3，4）
    a. 破坏请求和保持条件
        i. 进程运行前一次申请全部的资源
        ii. 只需要获得运行之初的资源就开始运行，逐步释放已用毕的资源，再请求新的资源

b．破坏不可抢占条件（可能导致饥饿）

i．当申请的资源得不到满足时，立即释放所有保持的所有资源

c．破坏环路等待条件

i．顺序资源分配法：编号递增顺序请求资源

2．避免死锁（银行家算法）

> 允许进程动态申请资源，但是会事先检查资源分配安全性
>
> 安全状态：按某种进程顺序为每个进程资源分配资源能够顺利完成
>
> 不安全状态并不一定会死锁，但安全状态一定不会死锁，避免死锁的实质就是设法避免不安全状态。

银行家算法

> m类资源，n个进程：可利用资源，最大需求，分配，需求
>
> - 设Requesti[j]=k,表示进程Pi需要k个Rj类型的资源
>   - （1）如果Requesti[j]<= Need[i,j],便转向步骤2；否则认为出错，因为它所请求的资源数已超过它所需要的最大值。
>   - （2）如果Requesti[j]<= Available[j],便转向步骤3；
>   否则，表示尚无足够资源，Pi需等待。
>   - （3）系统试探着把资源分配给进程Pi，并修改下面数据结构中数值
>   Available[j] = Available[j]- Requesti[j];
>   Allocation[i,j] = Allocation[i,j]+Requesti[j];
>   Need[i,j] = Need[i,j]-Requesti[j];
>   - （4）系统执行安全性算法，检查此次资源分配后系统是否处于安全状态以决定是否完成本次分配。若安全，则正式分配资源给进程Pi；否则，不分配资源，让进程Pi等待

3． 检测死锁

> 一种数据结构（资源分配图）
>
> 两种节点：进程节点，资源节点

> 两种边：请求边（进程节点→资源节点），分配边（资源节点→进程节点）

死锁检测算法

> 依次消除与不阻塞进程相连的边，直到无边可消
>
> 死锁定理：资源分配图不是完全可简化的说明发生了死锁

4．解除死锁

a．资源剥夺法

b．撤销进程法

c．进程回退法