《操作系统》期末速通教程

2. 进程管理

2.1 进程概述

[**多道程序并发运行**]

(1) 特性:

① 失去封闭性: 并发进程共享变量, 执行结果与运行速度有关.

② 间断性.

③ 不可再现性.

(2) 为保证并发程序能正确执行, 引入进程.

[**注**] 多道程序并发运行时, 进程间可能无关, 也可能有交互性.

[**进程**]

(1) 进程实体:

① 程序段: 能被调度程序调度到 CPU 上执行的代码段.

② 数据段: 进程使用的原始数据, 或产生中间数据和最终结果.

③ **进程控制块** (Process Control Block, PCB) .

(2) 进程定义: 进程是进程实体运行的的过程, 是系统进行资源分配和调度的独立单位.

(3) 特性:

① 动态性: 进程动态的创建、执行、消亡.

② 并发性.

③ 独立性.

④ 异步性: 进程间相互制约, 各自按独立的、不可预知的速度向前推进, 导致运行结果有不可再现性.

(4) 程序与进程的区别:

① 根本区别: 程序是静态的数据与指令的集合, 进程是动态的执行实体.

② 一个程序可产生多个进程.

③ 程序用于存储数据和接受系统启动, 进程用于并发执行与接受调度.

④ 程序存储需要存储器, 进程执行需要 CPU .

⑤ 程序永存, 进程有生命周期.

[**注 1**] 动态性是进程最基本的特性.

[**注 2**] 系统允许的最大进程数主要受内存大小限制.

[**注 3**] 父进程与子进程不共享虚拟地址空间.

[**进程的分类**]

(1) **系统进程**: 运行在**内核态**(又称**系统态**、**管态**)的进程.

(2) **用户进程**: 运行在**用户态**(又称**目态**)的进程.

[**注**] 系统进程可直接作 I/O 操作, 而用户进程不能.

[**进程的状态**]

(1) 三种基本状态:

① 就绪态: 已获得除 CPU 外的所有资源, 处于就绪队列.

② 执行态: 在 CPU 上运行.

③ 阻塞态: 等待某资源(不含 CPU)可用, 或等待 I/O 完成, 处于阻塞队列.

(2) 另外三种状态:

① 挂起态:

(i) 定义: 是一种不能立即投入运行的静止状态.

(ii) 分类: 静止就绪、静止阻塞.

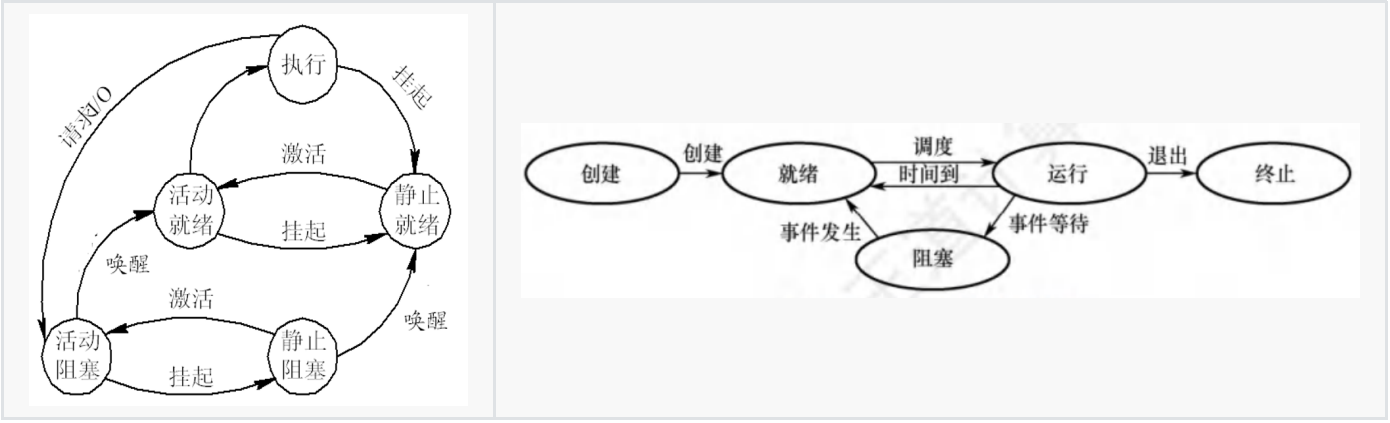
② 创建态:

(i) 定义: 进程正在被创建, 但仍未转到就绪态.

(ii) 特点: 有 PCB , 但未分配内存, 未进入调度队列.

③ 终止态.

(4) 示意图:



[**注 1**] 系统中未必有运行态的进程, 如单处理器系统死锁时, 所有进程都处于阻塞态.

[**注 2**] 一个进程可顺序地执行多个程序, 但同一时刻只能执行一个程序.

[**注 3**] 一个程序的一次执行可产生多个进程.

[**注 4**] 单处理器系统中, n 个进程至多 $(n-1)$ 个处于就绪态, 至多 n 个处于阻塞态.

[**注 5**] 一个进程释放了打印机, 可能会改变所有等待打印机的进程的状态.

[**注 6**] 分时 OS 中, 通常处于就绪态的进程最多, 因为阻塞发生频率不高.

[**注 7**] 一个进程从运行态变为就绪态 iff 时间片到, 此时一定发生进程调度.

[**注 7**] 进程被唤醒后优先级可能变大.

[**注 8**] 处于运行态的进程也可主动出让 CPU , 变为就绪态.

[**注 9**] 若系统中无运行进程, 则一定无就绪进程.

2.2 进程控制

[**PCB**]

(1) 进程创建时, OS 为之分配一个 PCB , 此后 PCB 常驻内存, 直至进程结束时回收.

(2) 组成:

① 进程描述信息: 进程标识符、用户标识符.

② 进程控制信息(资源分配清单): 内存或虚存的使用情况、打开文件列表、使用的 I/O 设备信息等.

③ 进程调度信息: 进程调度和对换有关的信息, 如进程状态、进程优先级等.

④ 处理机信息(CPU 上下文): CPU 中各寄存器的值.

(3) 作用:

① 是进程实体的一部分, 是进程存在的唯一标志.

② OS 通过 PCB 控制和管理进程.

[**原语**]

(1) 定义: 由若干条命令构成的、用于完成一定功能的代码段.

(2) 特点: 原子性, 即原语中的操作不可分割, 要么全做, 要么全不做.

(3) 应用: 是 OS 内核执行基本操作的主要方法.

[**进程创建**, **创建原语**].

(1) 过程:

① 为新进程分配一个唯一的进程标识号, 并申请一个空白的 PCB . 若 PCB 申请失败, 则进程创建失败.

② 为进程分配运行所需的资源, 如内存、文件、I/O设备、CPU 时间等. 这些资源或从 OS 获得, 或从父进程获得. 若资源不足, 则处于创建态, 等待内存资源.

③ 初始化 PCB , 主要为初始化标志信息、CPU 状态信息、CPU 控制信息, 并设置进程的优先级等.

④ 若进程就绪队列能接纳新进程, 则新进程入队, 等待调度.

(2) 引起进程创建的事件:

① 用户登录.

② 高级调度(作业调度): 从后备队列中选一个作业调入内存, 并为其创建进程.

③ 提供服务: 如 OS 响应用户请求时, 为用户创建一个子进程.

④ 应用请求: 如 Unix 系统的 fork() 函数.

[**进程终止**, **终止原语**]

(1) 过程:

① 根据被终止进程的进程标识符找到对应的 PCB , 读取进程状态.

② 若被终止进程处于运行态, 立即终止执行. 置调度标志为真, 在进程终止后重新调度.

③ 若被终止进程有子孙进程, 常需终止所有子孙进程.

④ 回收被终止进程的资源给父进程或 OS .

⑤ 将被终止进程的 PCB 从所在队列或链表中移除.

(2) 引起进程终止的事件:

① 正常结束: halt 、logoff .

② 异常结束: 越界错、保护错、非法指令、特权指令错、运行超时、等待超时、算术运算错、I/O 故障等.

③ 外界干预.

[**进程阻塞**]

(1) 进程执行过程中发生引发阻塞的事件, 无法继续执行, 则进程调用阻塞原语 block 将自身阻塞.

(2) 阻塞原语:

① 根据被阻塞进程的进程标识符找到对应的 PCB , 修改其状态为阻塞态.

② 保存现场, 将被阻塞进程的 CPU 状态保存到 PCB 中.

③ 将被阻塞进程的 PCB 插入对应事件的等待队列.

④ 重新调度.

[**进程唤醒**]

(1) 若进程引发阻塞的事件已满足(如 I/O 已完成), 则由相关进程(如用完并释放该 I/O 设备的进程)调用唤醒原语 wakeup , 唤醒等待事件的进程.

(2) 唤醒原语:

① 在事件的等待队列中找到被唤醒进程的 PCB .

② 将被唤醒进程的 PCB 移出等待队列.

③ 将被唤醒进程的 PCB 的状态置为就绪态, 并插入就绪队列,

[**注**] 进程从运行态变为阻塞态是主动行为, 从阻塞态变为就绪态是被动行为.

2.3 进程同步与互斥

2.3.1 进程同步与互斥

王道 p101 ~ 103

[**临界资源**]

(1) 定义: 一次仅允许一个进程使用的资源.

(2) 例: 多数物理设备(如打印机)、被若干进程共享的数据、公共队列等.

[**注 1**] **可重入代码**(又称**纯代码**, 不可修改)不是临界资源, 它在任意时刻可由多个进程共享.

[**注 2**] 共享程序段需用可重入代码编写.

[**临界区**] 进程中访问临界资源的代码段.

[**注**] 正在访问临界区的进程 P 因申请 I/O 而被中断时, 其它进程可抢占处理器, 但不可进入 P 的临界区.

[**例**] 系统中有 5 个并发进程涉及同一变量 A , 则 A 的临界区有 5 个, 即有 5 个操作 A 的代码段.

[**同步机制应遵循的规则**]

(1) 空闲让进.

(2) 忙则等待.

(3) 有限等待.

(4) 让权等待: 进程不能进入临界区时应释放 CPU , 不忙等. (原则上需遵循, 但非必须.)

[**记录型信号量**]

(1) 整型信号量的缺点: 忙等, 未遵循 "让权等待" .

(2) 记录型信号量的优点: 不忙等, 遵循 "让权等待" .

(3) **PV 操作**是低级进程通信原语, 不是系统调用.

```
void P(Semaphore S) {
    S.value--;
    if (S.value < 0) {
        将该进程插入信号量 S 的阻塞链表 S.L;
        block(S.L);
    }
}
```

```
void V(Semaphore S) {
    S.value++;
    if (S.value <= 0) {
        从 S.L 取一个被阻塞的进程 P;
        wakeup(P);
    }
}
```

(4) $S.value$ 的含义:

① > 0 时, 表示可用资源数.

② $= 0$ 时, 为临界状态, 无可用资源, 也无阻塞进程.

③ < 0 时, 表示阻塞进程数.

[**注**] 用 PV 操作实现进程同步时, 信号量的初值由用户指定, 未必为 0 .

[**例**] 有 4 个进程共享同一程序段, 每次允许 3 个进程进入该程序段.

用 PV 操作实现互斥时, 信号量取值范围为 $\{-1, 0, 1, 2, 3\}$.

[**AND 型信号量**]

(1) 策略: 临界资源分配采取原子操作, 要么全分配, 要么不分配.

(2) 优点: 预防死锁.

2.3.2 经典同步问题

[**生产者-消费者问题**]

(1) 定义与解决方法: 王道 p103 ~ 105 .

(2) 存在的唤醒:

① 生产者唤醒其它生产者.

② 生产者唤醒消费者.

③ 消费者唤醒其它消费者.

④ 消费者唤醒生产者.

[**读者-写者问题**] 定义与解决方法: 王道 p105 ~ 107 .

[**注**] 线程访问同一进程中的共享变量时才可能需互斥. 具体地, 读共享变量时无需互斥, 写共享变量时需互斥.

[**哲学家问题**]

(1) 定义与解决方法: 王道 p107 ~ 109 .

(2) 避免死锁的策略:

① 每位哲学家需同时拿左右两支筷子.

② 至多允许 4 位哲学家同时进餐.

③ 对哲学家依次编号, 奇数号哲学家需先拿左边的筷子, 偶数号哲学家先拿右边的筷子.

[**例**] 下面用 PV 操作解决哲学家问题. 在保证不死锁的前提下, 求信号量 $seat$ 的最大初值.

```
Semaphore chopsticks[5] = { 1, 1, 1, 1, 1 };
Semaphore seat;

Pi() {  // i 号哲学家
    ...
    P(seat);
    P(chopsticks[i]);
    P(chopsticks[(i + 1) % 5]);
    eat;
    V(chopsticks[i]);
    V(chopsticks[(i + 1) % 5]);
    V(seat);
}
```

[**答**] $seat$ 最大初值为 4 , 即同一时刻最多允许 4 位哲学家吃饭.

2.4 进程通信

[**进程通信**]

(1) 定义: 进程间的信息交换.

(2) 分类:

① 低级通信: 交换少信息量, 如信号量机制.

② 高级通信: 交换大信息量.

[**注 1**] 信号量机制实现低级通信的缺点:

① 效率低, 通信数据量少.

② 不透明, 用户需编程交换过程, 实现同步和互斥等.

③ OS 只提供内存空间.

[**注 2**] 信号量机制实现低级通信时, 信号量在内核或共享内存中, 不在 PCB 中.

[**高级进程通信**]

(1) 定义: 用户利用 OS 提供的通信命令进行进程间的信息交换.

(2) 特点: 高效传送大量数据.

(3) 分类:

① 共享存储: 信息无格式.

② 消息传递: 信息有格式.

③ 管道: 以文件方式.

[**注**] 高级通信中, 共享存储最快, 因为消息传递和管道都需要在用户空间和内核间作数据拷贝.

[**共享存储**]

(1) 定义: 通信的进程间存在一块可直接访问的内存空间, 通过对这块空间的读/写实现信息交换.

(2) 例: 生产者-消费者问题中的缓冲池.

(3) 缺点:

① 增加程序员负担.

② OS 只提供内存空间.

③ 低效, 只能传递少量数据.

[**注**] 两个进程共享空间需用系统调用实现, 而一个进程内的各线程本身就共享进程空间.

[**消息传递**]

(1) 定义: 进程间的数据交换以格式化的信息为单位.

(2) 分类:

① **直接通信**: 发送进程用 OS 提供的发送命令, 将消息发送给目标进程.

② **间接通信**(又称**信箱通信**): 以某种共享数据结构实体作为中介, 中介实体一般称为**信箱**.

[**管道**]

(1) **管道**是连接一个读进程和一个写进程的共享文件, 又称 **pipe 文件**.

(2) 特点:

① 只能由使用者进程创建.

② 只有管道的创建进程及其子孙进程能使用该管道.

③ 每次写入或读出的信息长度可变.

④ 每次数据交换单向, 即同一时刻只能有一个进程读, 一个进程写.

(3) 协调机制:

① 互斥: 不可同时读写管道.

② 同步: 写完才读, 读完才写.

③ 确认存在: 三方握手.

[**注**] 管道的容量一般是内存的一页.

2.5 线程

[**进程的不足**]

(1) 为实现多道程序并发运行, OS 需进行创建、撤销、切换进程的操作, 系统开销大.

(2) 进程同时作为资源分配单位和调度分派单位, 限制了并发程度的提高.

[**注**] 为解决进程的不足, 引入**线程**.

[**线程**]

(1) 定义: 轻量级进程, 是基本的 CPU 执行单元, 也是程序执行流的最小单元.

(2) 线程是进程中的一个实体, 一个进程可包含多个线程.

(3) 线程是被系统独立调度和分派的基本单位.

(4) 线程自身不拥有系统资源, 只有一点运行中必不可少的资源.

线程可与所处的进程的其它线程共享进程的所有资源.

(5) 优点:

① 减少程序并发时的开销.

② 提高系统并发性.

③ 便于进程通信.

(6) 缺点: 降低进程安全性. 一个进程的各线程共享进程空间, 一个进程出错可能影响整个进程.

[**注 1**] 线程包含 CPU 现场(CPU 中寄存器的值、堆栈指针等), 可独立执行程序.

[**注 2**] 线程的切换可能引起进程的切换.

[**注 3**] 线程的大小不超过所属进程的大小.

[**注 4**] 线程与所属进程不共享虚拟地址空间.

[**多线程 OS**]

(1) 特点:

① 多线程 OS 中, 进程作为拥有系统资源的基本单位.

② 进程常包含多个线程并为它们提供资源, 但进程不作为执行的实体, 进程的执行状态实质是其某个线程的执行.

(2) 优点:

① 速度快.

② 进程通信简便.

③ 设备并行性高.

[**注 1**] 无论系统中是否有线程, 进程都是拥有系统资源的基本单位.

[**注 2**] 多线程 OS 中, 系统认为线程是一种特殊的进程.

[**注 3**] 同一进程或不同进程中的线程都可并发执行.

[**注 4**] 键盘输入无需多线程, 系统只需用一个线程响应键盘输入.

2.6 死锁

[**死锁**]

(1) 定义: 多个进程因竞争资源形成僵局, 若无外力作用, 这些进程永远无法推进.

(2) 产生原因:

① 竞争不可剥夺的系统资源.

② 进程间推进顺序非法.

③ 信号量使用不当.

(3) 死锁的必要条件:

① 互斥条件: 请求的资源是临界资源.

② 请求并保持条件: 持有旧资源, 申请新资源.

③ 不可剥夺条件: 已获得的资源不可被外力剥夺, 只能主动释放.

④ 循环等待条件.

[**注 1**] 饥饿与死锁的区别与联系:

(1) 区别:

① 饥饿的进程只能有一个, 而死锁的进程至少两个.

② 饥饿的进程处于就绪态或阻塞态, 而死锁的进程一定处于阻塞态.

(2) 联系: 系统饥饿未必代表系统死锁, 但至少有一个进程的执行被无限推迟.

[**注 2**] 只有对不可剥夺的资源的竞争才会死锁, 对可剥夺的资源(如 CPU 、主存等)的竞争不会死锁.

[**注 3**] 满足死锁的必要条件未必发生死锁.

[**注 4**] 一个进程死循环, 可能饥饿, 但非死锁.

[**例**] 三个并发进程分别需要 $3, 4, 5$ 台同类设备, 为保证系统不死锁, 该设备最少需多少个.

[**解**] 由抽屉原理: 最少需 $(3-1)+(4-1)+(5-1)+1=10$ 个.

[**死锁处理**]

(1) 死锁预防: 设置限制条件, 破坏死锁的一个或多个必要条件.

(2) 死锁避免: 资源分配中加入检查, 防止系统进入不安全状态. 如银行家算法.

(3) 死锁检测与解除: 检测机构检测死锁, 通过剥夺资源或撤销进程等方式解除死锁.

[**死锁预防**]

(1) 破坏 "不可剥夺条件" :

① 策略: 已保持了某些不可剥夺的资源的进程请求新的资源不能满足时, 需释放已保持的资源.

② 缺点:

(i) 实现复杂.

(ii) 被剥夺资源的进程的工作全部作废, 代价高.

(iii) 资源反复申请和释放, 影响进程推进速度, 增加系统开销, 降低系统吞吐量.

③ 适用于状态易保存和恢复的资源, 如 CPU 的寄存器和内存资源; 不适用于打印机等资源.

(2) 破坏 "请求并保持条件" :

① 策略: 预先静态分配, 即进程在运行前一次性申请完所需的所有资源, 资源未满足时不投入运行.

② 优点:

(i) 实现简单.

(ii) 安全.

③ 缺点:

(i) 资源严重浪费.

(ii) 进程延迟执行.

(iii) 可能导致饥饿.

④ 为解决上述缺点, 改进: 允许进程只获得初期所需的资源后即开始运行, 运行过程逐步释放已分配且已使用完毕的全部资源后, 才能请求新资源.

(3) 破坏 "循环等待条件" :

① 策略: 对资源排序, 申请资源时需按规定次序进行.

② 缺点:

(i) 资源排序需相对稳定, 不利于增加新类型的设备.

(ii) 已获得的资源可能长期闲置.

(iii) 进程实际使用资源的顺序与编号次序不一致, 浪费资源.

(iv) 需按规定次序申请资源, 对用户编程有限制.

[**注 1**] 为进程设置优先级不能避免死锁.

[**注 2**] 信号量机制不能避免死锁, 反而可能加强 "互斥条件" 和 "请求并保持条件" .

[**注 3**] 死锁预防可保证系统一定不死锁.

[**例**] 哲学家问题中:

(1) 同时检查一位哲学家旁边的两支筷子是否都可用可预防死锁, 但浪费资源.

拿到筷子的人吃完后, 别人可继续吃, 不会导致饥饿.

(2) 限制允许拿筷子的哲学家数可预防死锁, 破坏了 "循环等待条件" .

(3) 给哲学家按顺序编号, 奇数号哲学家先拿左边的筷子, 偶数号哲学家先拿右边的筷子, 破坏了 "循环等待条件" .

[**例**] 若要求每个进程只能同时申请或拥有一个资源, 破坏了 "请求并保持条件" , 不死锁.

[**系统安全状态**] 若系统存在安全序列, 则称系统处于**安全状态**, 否则称系统处于**不安全状态**.

[**注**] 死锁状态一定是不安全状态, 不安全状态未必是死锁状态.

[**银行家算法**]

(1) 过程: 王道 p153 ~ 155 .

(2) 缺点:

① 很少有进程可在运行前知道所需的最大资源数.

② 系统内的进程数不固定.

③ 可用资源可能因故障突然变得不可用.

④ 开销大.

⑤ 实时性低.

[**注 1**] 银行家算法能避免死锁的原因: 任意时刻都能保证至少有一个进程可得到所需的全部资源.

[**注 2**] 银行家算法不能确定系统是否死锁.

[**例**] 设系统中有 m 个同类资源供 n 个进程共享, 每个进程至多申请 k $(k \geq 1)$ 个资源. 采用银行家算法分配资源, 为保证系统不死锁, 求各进程的最大需求量之和.

[**解**] 由银行家算法: 各进程申请的最大资源数 $\leq m$ 时存在安全序列.

极端情况: $(n-1)$ 个进程都申请 1 个资源, 剩下一个进程申请 m 个资源,

此时系统不死锁, 最大需求量 $= m + n - 1$.

[**死锁检测与解除**]

(1) 死锁检测:

① 资源分配图: 王道 p156 .

② 死锁定理: 资源分配图不可完全简化时, 系统死锁, 此时有边相连的进程即为死锁进程.

(2) 死锁解除:

① 剥夺资源: 剥夺部分死锁进程的资源.

② 撤销进程.

③ 回退进程.

[**注 1**] 死锁解除一般不会剥夺非死锁进程的资源.

[**注 2**] 若资源分配图中出现环路, 若每种资源只有一个, 则一定死锁.

[**注 3**] 资源分配图中, 若每个进程节点至少有一条请求边, 则是否死锁取决于资源是否充足.

[**注 4**] 死锁预防 $\rightarrow$ 死锁避免 $\rightarrow$ 死锁检测, 严格性降低, 并发性上升.

[**死锁公式**] 系统不发生死锁的条件: $\text{资源数} > \text{进程数} \times (\text{每个进程所需的最大资源数} - 1)$.

2.7 处理机调度

2.7.1 三级调度

[**三级调度**]

(1) 高级调度(作业调度): 外存 $\rightarrow$ 内存.

① 工作: 根据调度算法和计算机状态, 从外存上处于后备队列中的作业中选择一个或多个调入内存.

② 每个作业只调入和调出一次.

(2) 中级调度(内存调度): 外存 $\leftrightarrow$ 内存.

① 工作: 将暂不能运行的进程调至外存等待, 即挂起. 实际是内存管理中的对换.

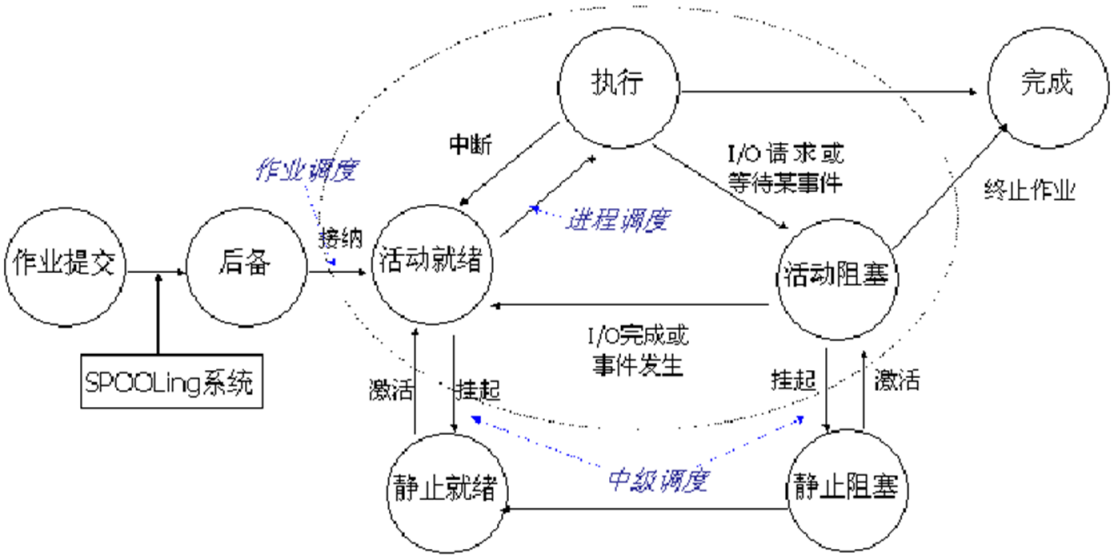
② 目的: 提高内存利用率和系统吞吐量.

(3) 低级调度(进程调度): 内存中.

① 工作: 根据调度算法, 从就绪队列中选择一个进程, 为之分配 CPU .

② 功能: 保存现场, 调度新程序, 新程序获取 CPU 控制权.

(4) 三级调度间的联系:

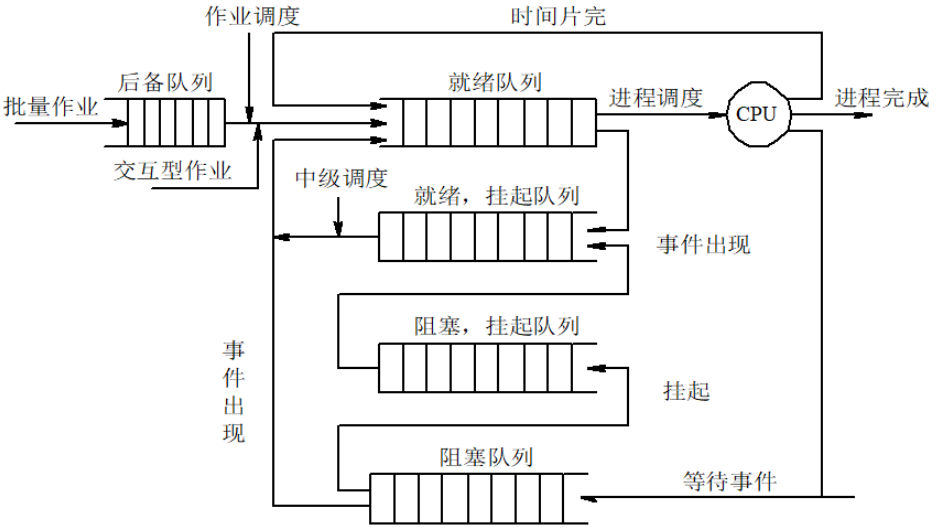


(5) 调度对象:

① 高级调度: 作业.

② 中级调度、低级调度: 进程.

(6) 三级调度队列:



[注1] 多道批处理系统大多配有作业调度, 其它系统常无需配有作业调度.

[注 2] 进程调度是最基本的调度, 各种 OS 都需配置进程调度.

[注 3] 高级调度决定哪些进程可进入系统处理, 控制了并发度.

[注 4] 作业与进程的区别:

① 作业是用户提交的, 进程是系统生成的.

② 作业以用户任务为单位, 进程以系统控制为单位.

[**进程调度**]

(1) 进程调度需要上下文切换, 其中进程的**上下文**指进程的代码、数据和支持进程执行的所有运行环境, 不涉及主存和外存的数据交换.

(2) 发生进程调度的时机:

① 正在运行的进程运行完毕.

② 正在运行的进程自我阻塞.

③ 正在运行的进程时间片用完.

④ 正在运行的进程出错.

[**注 1**] 系统开始调度时也是一次调度. 若考虑进程调度的开销, 则也需考虑刚开始进程调度时的开销.

[**注 2**] 进程调度不涉及阻塞队列中的进程, 阻塞的进程需进入就绪队列后才能被调度.

[**进程调度方式**]

(1) **非抢占方式(非剥夺方式)**: 正在运行的进程运行完毕或阻塞, 才让出 CPU .

① 优点:

(i) 实现简单.

(ii) 系统开销小.

② 缺点:

(i) 难满足紧急任务要求.

(ii) 不适用于实时系统.

③ 适用于早期的批处理系统, 不适用于分时系统和实时系统.

(2) **抢占方式(剥夺方式)**: 运行暂停正在运行的进程, 将 CPU 重分配给另一进程.

① 优点:

(i) 提高系统吞吐量.

(ii) 减少系统响应时间.

② 原则: 优先权原则、短作业优先原则、时间片原则.

2.7.2 调度算法的指标

[**CPU 利用率**] $\text{CPU 利用率} = \dfrac{\text{CPU 有效工作时间}}{\text{CPU 有效工作时间} + \text{CPU 空闲等待时间}}$.

[**系统吞吐量**]

(1) 定义: 单位时间内 CPU 完成的作业数.

(2) 影响因素:

① 作业长短:

(i) 长作业消耗 CPU 时间长, 系统吞吐量低.

(ii) 短作业消耗 CPU 时间短, 系统吞吐量高.

② 调度算法.

[**周转时间**]

(1) 周转时间: 作业从提交到结束经历的时间.

① $\text{周转时间} = \text{作业完成时间} - \text{作业提交时间}$.

② $\text{平均周转时间} = \dfrac{1}{n}\sum_{i=1}^{n} \text{作业 } i \text{ 的周转时间}$.

(2) 带权周转时间: 作业周转时间与作业实际运行时间之比.

① $\text{带权周转时间} = \dfrac{\text{作业周转时间}}{\text{作业实际运行时间}}$.

② $\text{平均带权周转时间} = \dfrac{1}{n}\sum_{i=1}^{n} \text{作业 } i \text{ 的带权周转时间}$.

[**等待时间**]

(1) 定义: 进程处于等待 CPU 的时间之和.

(2) 等待时间越长, 用户满意度越低.

(3) 调度算法不影响作业的执行时间和 I/O 时间, 只影响作业在就绪队列中的等待时间.

(4) 等待时间是调度算法的评价指标之一.

[**响应时间**]

(1) 定义: 用户提交请求到系统首次响应的时间.

(2) 交互式系统中, 一般用响应时间衡量调度算法.

2.7.3 调度算法

王道 p70 ~ 74

[**调度算法适用对象**]

(1) 作业、进程:

① FCFS .

② SJF .

③ 优先级调度.

④ 多级反馈队列.

(2) 作业: 高响应比优先调度.

(3) 进程: RR .

[**注 1**] 单处理器系统中, 进程何时占用处理器和占用时间长短由进程特点和调度策略共同决定.

[**注 2**] 进程调度的目标是: 让进程轮流使用处理器.

[**注 3**] 处于临界区的进程在退出临界区前可能也会被调度.

[**先来先服务算法**, **First Come First Service**, **FCFS**]

(1) 特点: 非抢占式.

(2) 优点:

① 公平.

② 实现简单.

③ 算法开销小.

④ 利于长作业.

⑤ 利于 CPU 繁忙型作业.

(3) 缺点:

① 不利于短作业.

② 不利于 I/O 繁忙型作业.

[**短作业优先算法**, **Shortest Job First**, **SJF**]

(1) 特点: 非抢占式或抢占式.

(2) 优点:

① 利于短作业.

② 平均等待时间、平均周转时间最短.

(3) 缺点:

① 不利于长作业, 增加周转时间.

② 长作业可能饥饿.

③ 不能保证实时性.

④ 作业的执行时间基于用户估算, 准确性不足.

(4) 适用于批处理系统.

[**优先级调度算法**]

(1) 优先级调度算法分类:

① 非抢占型.

② 抢占型.

(2) 缺点: 可能饥饿.

(3) 优先级类型:

① 静态优先级: 创建进程时确定, 运行过程中不变.

② 动态优先级: 随进程推进或等待时间增加而改变.

(4) 优先级设置原则:

① 系统进程 > 用户进程.

② 交互型进程 > 非交互型进程, 即前台进程 > 后台进程.

③ I/O 密集型进程 > CPU 密集型进程.

(5) 优先级变化时机:

① 优先级降低: 进程时间片用完.

② 优先级上升: 进程从阻塞态变为就绪态、进程长期处于就绪队列.

(6) 适用于实时 OS .

[**高响应比优先调度算法**]

(1) 响应比 $= \dfrac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}} = \dfrac{\text{响应时间}}{\text{要求服务时间}}$.

(2) 优点:

① 兼顾长短作业:

(i) 等待时间相同时, 要求服务时间越短, 响应比越高, 利于短作业. 类似于 SJF .

(ii) 要求服务时间相同时, 等待时间越长, 响应比越高. 类似于 FCFS .

② 长作业的响应比随等待时间的增加而上升, 等待时间足够长时能保证获得 CPU , 不饥饿.

(3) 缺点: 每次调度前都需计算响应比, 系统开销大.

(4) 适用于分时 OS .

[**时间片轮转**, **Round Robin**, **RR**]

(1) 优点:

① 兼顾长短作业.

② 响应时间短.

(2) 缺点:

① 系统吞吐量和平均周转时间不如批处理系统.

② 上下文切换开销大.

(3) 适用于分时 OS .

(4) 时间片过大时, 退化为 FCFS 算法.

[**注**] RR 算法是绝对可抢占的.

[**多级反馈队列调度算法**]

(1) 优点:

① 兼顾长短作业.

② 响应时间短.

③ 可行性高.

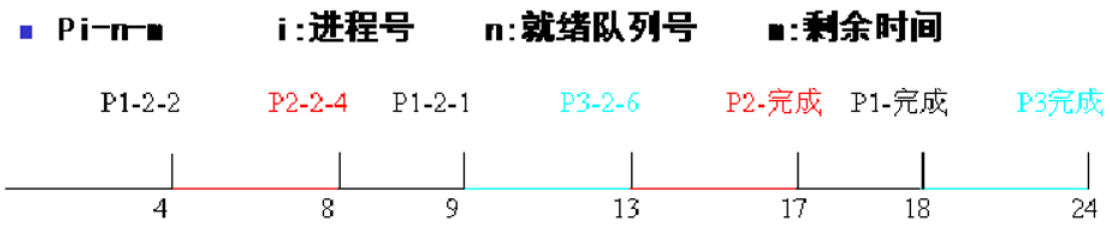
(2) 缺点: 实现复杂.

(3) 适用于分时 OS .

[例] 某 OS 采用三级反馈队列调度, 时间片大小分别为 4 、8 、16 . 现有三个进程, 到达时刻和运行时间如下. 写出调度顺序.

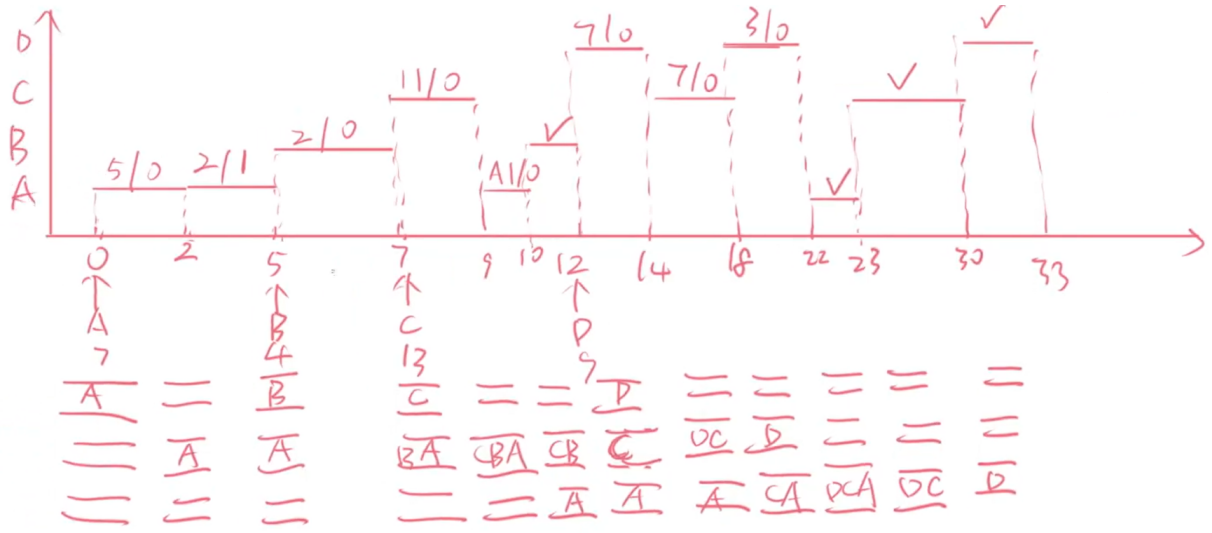
进程	到达时刻	运动时间
P_1	0	6
P_2	2	8
P_3	9	10

[解]



[例] 某 OS 采用三级反馈队列调度, 时间片大小分别为 2 、4 、8 . 四个进程 A 、B 、C 、D 的到达时间分别为 0 、5 、7 、12 , 运行时间分别为 7 、4 、13 、9 . 求每个进程的带权周转时间.

[解] 调度过程如下图所示:



进程	到达时间	运行时间	完成时间	周转时间	带权周转时间
A	0	7	23	23	$\frac{23}{7}$
B	5	4	12	7	$\frac{7}{4}$
C	7	13	30	23	$\frac{23}{13}$
D	12	9	33	21	$\frac{7}{3}$

2.7.4 实时调度

[实时任务]

(1) 实时任务: 需在截止时间前开始或结束的任务.

(2) 开始截止时间: 需在指定时间前开始.

(3) 完成截止时间: 需在指定时间前完成.

[实现实时调度的条件]

(1) 提供必要信息:

① 就绪时间.

② 开始截止时间、完成截止时间.

③ 处理时间.

④ 资源要求.

⑤ 优先级.

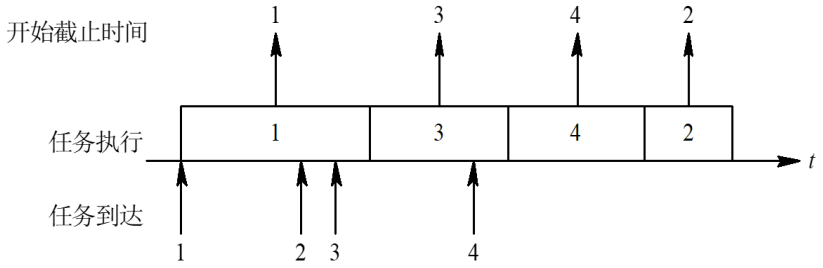
(2) 系统处理能力强.

设 N 处理机系统中有 m 个周期性硬实时任务, 其中 $i\ \ (1 \leq i \leq m)$ 号任务的处理时间为 C_i , 周期时间为 P_i , 则系统可调度 iff $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq N$.

(3) 采用抢占式调度. 若已知任务的开始截止时间, 则也可采用非抢占式调度.

(4) 有快速切换机制.

[最早截止时间优先算法, Earliest Deadline First, EDF]

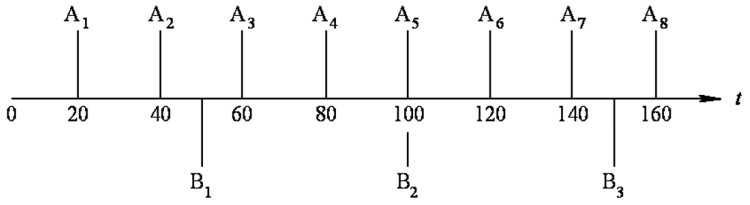


[最低松弛度优先算法, Least Laxity First, LLF]

(1) 松弛度 $=$ 完成截止时间 $-$ 剩余处理时间 $-$ 当前时间 .

(2) 松弛度为 0 时称为**临界状态**, 发生抢占.

[**例**] 某实时系统中有两个周期性实时任务 A 和 B，其中 A 每 20 ms 执行一次，执行时间为 10 ms；B 每 50 ms 执行一次，执行时间为 25 ms。采用 LLF 算法，写出 $0 \sim 80$ ms 的调度顺序。



[**解**]

(1) 任务的临界时刻:

① $A_1: 20-10=10$，$A_2: 40-10=30$，$\cdots$，

则 A 在 $0 \sim 80$ ms 的临界时刻为 $10, 30, 50, 70$。

② $B_1=50-25=25$，$B_2: 100-25=75$，$\cdots$，

则 B 在 $0 \sim 80$ ms 的临界时刻为 $25, 75$。

(2) 调度顺序:

时刻	松弛度	执行任务
T=0	A1=20-10-0=10，B1=50-25-0=25	00-10：A1完成
T=10	A2到T=20才开始，B1直接运行	10-30：B1余5
T=30	A2=40-10-30=0，B1=50-5-30=15 A2抢占B1	30-40：A2完成
T=40	A3=60-10-40=10，B1=50-5-40=5	40-45：B1完成
T=45	B2到T=50才开始，A3直接运行	45-55：A3完成
T=55	A4到T=60才开始，B2直接运行	55-70：B2余10
T=70	A4=80-10-70=0，B2=100-10-70=20 A4抢占B2	70-80：A4完成
T=80	A5=100-10-80=10，B2=100-10-80=10	80-90：B2完成
……	……	……