进程管理

进程概念的引入

1. 程序的执行顺序
    1. 串行执行

       编写程序简单方便，资源独占性导致资源利用率非常低下

        - 封闭性 : 程序执行的时候独占全机的资源，运行环境封闭(只能由该程序改变资源的状态)
        - 可再现性 : 程序可再现性(程序的运行不随机，结果可预测)
    2. 并行执行

       程序的控制和编写困难,以资源共享为条件，但是资源利用率高，提高计算机系统的处理能力

        - 增强计算机系统的处理能力和资源利用率
        - 失去封闭性和可再现性，出现有关时间的错误
        - 进程之间出现了制约关系
            - 间接制约 : 互斥，多进程抢占资源
            - 直接制约 : 协作同步
        - 程序与CPU的活动并不是一一对应
            - 前者是静态概念，可以长期存储的
            - 后者是动态概念，存在生命周期，临时的
2. 进程概念
    1. 定义

       描述系统中的并发活动引入的概念，进程又叫做任务，进程(任务)是操作系统中最基本和重要的概念

       动态特性

        - 动态性 : 程序的一次执行过程，临时的，有生命周期的
        - 独立性 : 系统资源分配和调度的独立单位
        - 并发性 : 程序之间可以并发执行
        - 结构性
            - 操作系统为进程管理方便创建**PCB,**记录进程的属性信息
            - 进程组成部分
                - 程序 : 多个
                - 数据 : 多个
                - PCB : 一个

进程的描述

1. PCB
    - 进程的唯一标识
    - 包含进程的描述信息和管理进程的控制信息
    - 包含的基本信息
        - 进程标识数(PID) : 系统的每一个进程都有一个唯一的标识，区分不同的进程
        - 进程的状态，调度，存储器管理信息(PCB在内存还是外存)
        - 进程使用资源信息 : 进程的I/O,进程的正在打开文件信息
        - CPU现场保护区 : 当程序运行出现问题或者中断的时候我们需要将进程的运行现场保存

        - PC计数器
        - 程序状态字 : 00 (核心态), 11
        - 通用寄存器
    - 记账信息(账号，CPU使用的时间量-可能之后在调度方面有作用)
    - 进程之间的家族关系
    - 进程连接指针 : 捆绑相同状态的PCB
2. 进程状态(记录在PCB中)
    - 创建态 : 进程创建初始化(大多数的操作系统都存在可以创建的进程的数目限制)
    - 终止态 : 进程死亡(尚未小时，等待其他进程收集相关的信息)
    - 就绪态 : 获得了除了CPU以外的其他全部资源，等待系统分配CPU
    - 运行态 : 正在CPU上执行的进程状态，一个CPU上某一时刻只能有一个进程运行
    - 阻塞态 : 进程等待事件的发生，即使CPU空闲也不可以运行
3. 进程的组织
    1. 线性表 : PCB组织成数组快速访问
        1. 简单，节省空间
        2. 系统开销大(查找对应的PCB)
    2. 链接表
        1. 将属于同一个状态的进程连接成一个队列，多种队列(就绪队列，阻塞队列,...)
        2. 处于不同的队列中的进程又可以根据调度策略组织成多种队列
            1. 就绪 : 调度策略决定多个就绪队列
            2. 阻塞 : 阻塞原因决定多个阻塞队列
        3. 系统可以设置一个指向当前正在运行的进程的指针,相对于线性结构来说加快查询当前的进程

进程控制

- 系统使用一些具有特定功能的程序段创建和撤销进程以及完成进程个状态之间的转换的一系列有效管理
- 进程控制程序 : 内核实现，原语一级操作不可中断
- 进程控制原语
    - 创建原语
        - 扫描PCB表，找到一个空闲的PCB，创建进程的标识
        - 如果进程的实体不在内存中，分配空间调入主存
        - 创建参数用来更新PCB(初始化PCB)
        - 设置为就绪状态加入就绪队列中
    - 撤销原语

      进程已经完成指定任务或者不可以继续运行，应该撤销系统
        - 在PCB表中寻找对应的PCB块
        - 考虑子进程一并撤销
        - 进程的占用空间归还系统
        - 最后撤销PCB
    - 阻塞原语
        - 将进程自己从运行态变成阻塞态
        - 中断正在处于运行态的CPU
        - 保存现场在PCB的CPU现场保护区中
        - 状态设置为阻塞态，进入相应的等待队列
        - 转处理机调度
    - 唤醒原语
        - 中断来临(另一个进程的发送消息)查询对应的进程是否存在
        - 存在唤醒对应的进程，转就绪态，进入就绪队列，结束中断处理
    - 挂起原语 / 解挂

- 只有处于活动状态的进程才可以被处理机进行调度
- 分时系统，把进程从内存换到外存，进程就处于静止状态，不被调度
- 挂起 : 活动 -> 静止 / 解挂 : 静止 -> 活动

处理机调度

1. 系统中的进程数目非常多(用户进程 + 系统进程)
2. 处理机调度级别
    1. 高级调度 : 作业调度，分时系统中没有
    2. 低级调度 : 进程调度,使用某种调度策略
        1. 目的让一些进程可以并发执行，最大化CPU利用率，进程调度比较频繁
        2. 功能，方式，时机，算法
    3. 中级调度 : 交换调度

        将处于主存就绪或者主存阻塞暂时不具备运行条件的进程换出道外存交换区，并将外存中具有运行条件的进程换入主存
3. 进程调度的功能
    1. 管理系统中每个进程的执行状况 : 状况信息在PCB中
    2. 选择进程真正占有CPU
    3. 进程的上下文切换
        - 进程上下文(保存在PCB中)

            操作系统为运行进程而设置的相应的运行环境，物理实体
            1. 用户级上下文
                - 程序
                - 数据
                - 用户栈
            2. 寄存器级上下文

                CPU现场信息
            3. 系统级上下文
                - PCB
                - 核心栈
        - 切换

            将正在执行的进程的上下文保存在进程的PCB中，以便恢复使用，将选中的进程的运行现场回复并将CPU控制权交还给选中进程使其执行
4. 进程调度方式和时机
    1. 调度方式
        1. 非抢先式 : 简单系统开销小，批处理系统
        2. 抢先式 : 分时，实时系统

            抢先下我们可以基于某一种调度策略剥夺CPU给其他的进程
            - 时间片轮转
            - 优先级调度
    2. 调度时机
        - 自身
            - 进程完成或者终止
            - I/O请求
            - 进程执行原语操作
        - 系统

        - 时间片轮转
        - 优先级调度

5. 调度算法
    1. 作业调度算法
        - 先来先服务(FCFS)
            - 节省机器时间，运行效率高
            - 容易被大作业进程垄断，平均等待时间延长
        - 最短作业进程优先调度(SJF)
            - 对于运行时间短的进程有利，长时间作业会出现饿死现象
            - 平均等待时间和周转时间最佳
        - 响应比高者优先算法(HRN)
            - 响应比定义 : $R_p = 1 + \frac{Wait}{Run}$

              作业等待时间和作业估计运行时间
    2. 进程调度算法
        - 优先级调度算法

          CPU分配给就绪队列中优先级最高的进程
            - 静态优先级 :
                - 进程创建的时候建立
                - 系统进程高优先级
                - 用户进程视情况而定(资源申请情况)
                - 申请资源少高优先级
                - 优先级分配不可变动
                - 对低优先级进程不公平
            - 动态优先级
                - 优先级不断的动态调整
                - 动态调整优先级需要计算开销
                - 通常根据进程占用的**CPU**时间长短或者等待**CPU**的时间长短动态的调整
        - 轮转法(RR)
            - 多用于分时系统中
            - 定时发出中断，然后进入处理机调度程序
            - 按照顺序依次轮转执行时间片
            - 时间片长短的确定原则 : 既要保证系统各个用户进程及时地得到响应，又不要因时间片太短而增加调度的开销，降低效率。
        - 多级反馈队列轮转法
            - 系统可以设置多个就绪队列，进程在生命周期中可以在多个队列中存在
            - 队列可以采用前后台运行，前台使用轮转法，后台使用先来先服务法
            - 前台进程的优先级高于后台的进程
            - 前台的队列的时间片长度也不一定相同，高优先级队列进程的时间片短**(**平衡的考虑**)**
            - 总是先调度优先级高的队列，只有当该队列为空才会执行较低优先级队列的进程
            - 只要高优先级队列中存在进程，立刻转处理机调度
            - 运行**2~3**个时间片还未完成的进程降级
        - 实时系统方法
            - 时钟驱动法 : 每个任务的调度安排都是在系统开始运行前就确定
            - 加权轮转法 : 轮转的时间片可以不同

线程

1. 为了减少程序并发执行系统付出的时间和空间开销，引入线程模型
    1. 进程是一个逻辑上的作业，线程是子任务

2. 线程是进程中一个可执行实体，操作系统调度的独立单位
3. 一个进程可以存在多个线程，多线程共享进程的所有资源(互斥访问)
4. 进程是分配资源的基本单位，线程是调度执行的基本单位

2. 线程控制块

   管理线程的全部信息

   1. 唯一标识
   2. CPU现场信息(处理机状态和运行现场的寄存器组)
   3. 两个堆栈 : 用户态和核心态传递参数
   4. 独立的**PC**计数器，私有存储区
   5. 进程和线程指针

3. 线程消耗资源少(私有存储区)，轻量级进程

4. 进程和线程的比较

   - 资源 : 线程消耗资源很少
   - 调度(线程的优点) : 进程上下文切换代价高，线程只需交换一小部分资源，提高系统的效率(快)
   - 并发性 : 线程并发性提高
   - 安全性(线程缺点) : 多进程共享资源，线程组公用的资源被恶意修改导致错误

5. 线程分类

   1. 用户级线程

      - 用户程序利用用户态运行的线程库创建，内核不知道
      - 内核仍然以进程为单位调度，用户级线程被阻塞，对应进程也阻塞
      - 进程内部存在线程表**(**运行时系统 **:** 管理线程的过程集合**(**基本的原语**,**每个进程都有**))**，内核中只存有进程表

   2. 核心级线程

      - 内核实现有关线程的所有管理工作,系统调用实现
      - 内核中既有进程表也有线程表
      - 线程的阻塞对线程组其他线程和对应进程没有影响，线程在内核的线程表中存在记录(用户级没有记录)
      - 用户对内核发生系统调用创建核心级线程
      - **P.S :** 内核线程(操作系统内核本身) != 核心级线程
      - 线程调度程序 : 无论线程属于哪一个进程，都是按照线程优先级分队列调度(内核线程表)

   3. 组合

      - 两者都支持
      - 用户级多个线程对应核心级多个线程
      - 并行性提高(宏观和微观)，加快运行速度