进程、线程

并发与并行

进程

进程的概念

我们编写的代码只是一个存储在硬盘的静态文件，通过编译后就会生成二进制文件，当我们运行这个可执行文件后，它会被装载到内存中，接着CPU会执行程序中的每一条执行，这个「运行中的程序」被称为「进程」。

进程的状态

活动期间

- 运行状态（ Runing ）
  - 该时刻进程占用CPU
- 就绪状态（ Ready ）
  - 可运行，但因为其他进程正在运行而暂时停止
- 阻塞状态（ Blocked ）
  - 该进程正在等待某一事件发生（例如等待输入/输出操作的完成）而暂时停止运行，这时，即使给它CPU控制权，它也无法运行

其他

- 创建状态（ new ）
  - 进程正在被创建的状态
- 结束状态（ Exit ）
  - 进程正在从系统中消失时的状态

状态变迁

- NULL->创建状态
  - 一个新进程被创建时的第一个状态
- 创建状态->就绪状态
  - 当进程被创建完成并初始化后，一切就绪准备运行时，变为就绪态（过程非常快）
- 就绪态->运行状态
  - 处于就绪状态的进行被操作系统的进程调度器选中后，就分配给CPU正式运行该进程
- 运行状态->结束状态
  - 当进程已经运行完成或出错时，会被操作系统作结束状态处理
- 运行状态->就绪状态
  - 处于运行状态的进程在运行过程中，由于分配给它的运行时间片用完，操作系统会把该进程变为就绪态，接着从就绪态选中另一个进程运行
- 运行状态->阻塞状态
  - 当进程请求某个事件且必须等待时，例如请求I/O事件
- 阻塞状态->就绪状态
  - 当进程要等待的时间完成时，它从阻塞状态变到就绪状态
- 挂起
  - 阻塞挂起状态
    - 进程在外存（硬盘）并等待某个事件的出现
  - 就绪挂起状态
    - 进程在外存（硬盘），但只要进入内存，即刻立即运行

进程的控制结构

在操作系统中，是用「进程控制块（ process control block , PCB ）」数据结构来描述进程的

PCB是进程存在的唯一标识，如果进程消失，PCB也会随之消失

PCB具体包含什么信息？

- 进程描述信息
  - 进程标识符：标识各个进程，每个进程都有一个并且唯一的标识符
  - 用户标识符：进程归属的用户，用户标识符主要为共享和保护服务
- 进程控制和管理信息
  - 进程当前状态：如new，ready，running，waiting或blocked等
  - 进程优先级：进程抢占CPU时的优先级
- 资源分配清单
  - 有关内存地址空间或虚拟地址空间的信息，所打开文件的列表和所使用的I/O设备信息
- CPU相关信息
  - CPU中各个寄存器的值，当进程被切换，CPU的状态信息都会被保存在相应的PCB中，以使进程重新执行时，能从断点处继续执行

每个PCB是如何组织的？

通常是通过链表的方式进行组织，把具有「相同状态的进程链在一起，组成各种队列」

- 将所有处于就绪状态的进程链在一起，称为「就绪队列」
- 把所有因等待某时间而处于等待状态的进程链在一起就组成了各种「阻塞队列」
- CPU运行队列在单核CPU系统中则只有一个运行指针，因为单核CPU在某个时间，只能运行一个程序

进程的控制

创建进程

操作系统允许一个进程创建另一个进程，而且允许子进程继承父进程所有的资源。当子进程被终止，其在父进程继承的资源应当还给父进程。当父进程被终止，其所有子进程也将被终止

- 流程
  - 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，PCB是有限的，如果申请失败则创建失败
  - 为进程分配资源，此处如果资源不足，进程就会进入等待状态，以等待资源
  - 初始化PCB
  - 如果进程的调度队列能够接纳新的进程，那就将进程插入到就绪队列，等待被调度运行

终止进程

进程可以有3种终止方式：正常结束、异常结束、以及外界干预（ kill ）

- 流程
  - 查找需要终止的进程PCB
  - 如果处于执行状态，则立即终止该进程的执行，让后将CPU资源分配给其他进程
  - 如果其还有子进程，则应该将其所有的子进程终止
  - 将该进程 所拥有的的全部资源归还给父进程或操作系统
  - 将其PCB所在队列中删除

阻塞进程

当进程需要等待某一事件完成时，它可以调用阻塞语句把自己阻塞等待。而一旦被阻塞等待，它只能由另一个进程唤醒

- 流程
  - 找到将要被阻塞进程标识号对应的PCB
  - 如果该进程为运行状态，则保护其现场，将其转化为阻塞状态，停止运行
  - 将该PCB插入到阻塞队列中

唤醒进程

进程由「运行」转化为「阻塞」状态是由于进程必须等待某一事件的完成，所以处于阻塞状态的进程是绝对不能唤醒自己。只有当进程所期待的时间出现时，才由发现者进程将睡眠态唤醒

进程的阻塞和唤醒是一对功能相反的语句，如果某个进程调用了阻塞语句，则必有一个与之对应的唤醒语句。

- 流程
  - 在该事件的阻塞队列中找到相应的PCB
  - 将其从阻塞队列中移出，并将其状态设置为就绪
  - 将PCB插入到就绪队列中，等待程序调度

进程的上下文切换

从一个进程切换到另一个进程运行，称为进程的上下文切换

CPU上下文切换

CPU上下文切换是把前一个任务的CPU上下文（CPU寄存器和程序计数器）保存起来，然后加载新任务的上下文到这些寄存器和程序计数器，最后再跳转到程序计数器所指的新位置，运行新任务

系统内核会存储保持下来的上下文信息，当此任务再次被分配给CPU运行时，CPU会重新加载这些上下文，这样就能保证原来的任务状态不受影响

「任务」主要包含进程、线程、和中断，可以根据任务的不同，把CPU上下文切换成「进程上下文切换、线程上下文切换和中断上下文切换」

进程的上下文切换

进程的上下文切换到底是切换什么？

进程的上下文切换不仅包含了虚拟内存、栈、全局变量等用户空间的资源，还包括了内核堆栈、寄存器等内核空间的资源

通常会把交换的信息保存在进程的PCB，当要运行另外一个进程的时候，我们需要从这个进程的PCB取出上下文，恢复到CPU，使得这个进程可以继续执行

线程

为什么使用线程？

- 解决实体之间可以并发
- 解决时间之间共享相同的地址空间

线程与进程的比较

什么是线程？

线程是进程当中的一条执行流程

现场的优缺点

- 优点
  - 一个进程中可以同时存在多个进程
  - 各个线程之间可以并发执行
  - 各个线程之间可以共享地址空间和文件等资源
- 缺点
  - 当进程中的一个线程崩溃，会导致其所属进程的所有线程崩溃

具体

- 进程是资源（ 包括内存、打开的文件等 ）分配的单位，线程是CPU调度的单位
- 进程拥有一个完整的资源平台，而线程只独享必不可少的资源，如寄存器和栈
- 线程同样具有就绪、阻塞、执行三种基本状态，同样具有状态之间的转换关系
- 线程能减少并发执行的时间和空间开销

性能

- 线程的创建的时间比进程快，因为进程在创建的过程中，还需要资源管理信息，比如内存管理信息、文件管理信息，而线程在创建的过程中，不会涉及这些资源，而是共享进程的资源
- 线程的终止时间比进程快，因为线程释放的资源相比进程少很多
- 同一个进程内的线程切换比进程快，因为线程具有相同的地址空间（ 虚拟内存共享 ），这意味着同一个进程的线程都具有同一个页表，那么在切换的时候不需要切换页表。相对于进程之间的切换，切换的时候要把页表给换掉，而页表切换过程开销是比较大的。
- 由于同一进程的各个线程间共享内存和文件资源，那么在线程之间数据传递的时候，就不需要经过内核了，这使得线程之间的数据交互效率更高了

线程的上下文切换

- 当进程只有一个线程时，可以认为进程就等于线程
- 当进程拥有多个线程时，这些线程会共享相同的虚拟内存和全局等资源，这些资源在上下文切换的时候不需要修改的
- 线程也有自己的私有数据，比如栈的寄存器等，这些在上下文切换的时也需要保存
- 如果两个线程不是属于同一个进程，切换的过程就跟进程上下文切换一样
- 当两个线程是属于同一个进程，因为虚拟内存是共享的，在切换时，虚拟内存这些资源就保持不同，只需要切换线程的私有数据、寄存器等不共享数据

线程的实现

- 用户线程（ User Thread ）
  - 在用户空间实现的线程，不是由内核管理的线程，是用户态的线程库来完成线程的管理
- 内核线程（ Kernel Thread ）
  - 在内核中实现的线程，是由内核管理的线程
- 轻量级线程（ LightWeight Process ）
  - 在内核中来支持用户线程

调度

调度时机

非抢占式调度算法

挑选一个进程，然后让该进程运行直到被阻塞，或者该进程退出，才会调用另外一个进程，完全不理会时钟中断

抢占式调度算法

挑选一个进程，然后让该进程只运行某段时间，如果在该时段结束时，该进程仍然在运行，则会挂起，接着调度程序从就绪队列挑选另外一个进程。这种抢占式调度处理，需要在时间间隔的末端发生「时钟中断」，以便CPU控制返回给调度程序进行调度，也就是常说的「时间片机制」

调度原则

「CPU利用率」调度程序应确保CPU是始终匆忙的状态，可以提高CPU的利用率

「系统吞吐量」吞吐量表示的是单位时间内CPU完成进程的数量，长作业的进程会占用较长的CPU资源，因此会降低吞吐量，相反，短作业的进程会提升系统吞吐量

「周转时间」周转时间是进程运行和阻塞时间总和，一个进程的中周转时间最小越好

「等待时间」等待时间不是阻塞状态的时间，而是进程处于就绪态队列的时间，等待的时间越长，用户越不满意

「响应时间」用户提交请求到系统第一次产生响应所花费的时间，在交互式系统中，响应时间是衡量调度算法好坏的主要标准

调度算法

- 先来先服务（ First Come First Sevred, FCFS ）算法（非抢占）
- 最短作业优先（ Shortest Job First, SJF ）调度算法
- 高响应比优先（ Highest Response Ratio Next, HRRN ）调度算法
- 时间片轮转（ Round Robin, RR ）调度算法
- 最高优先级（ Highest Priority First，HPF ）调度算法
- 多级反馈队列（ Multilevel Feedback Queue ）调度算法