进程通信、同步与调度

江学强，PB16120100

1. 进程间通信

有些进程在执行过程中可能会影响其他进程的执行同时也可能被其他进程影响，也就是说这些进程互相不独立，所以需要实现进程间通信的机制。此外实现进程间通信可以加速一些问题的处理、使系统的模块化程度更高等。实现进程间通信主要有两种模型：内存共享模型和数据传输模型，内存共享模型是指一块内存可以被一系列合作的进程共享，信息传递模型是指在进程间互相传递必要的信息。

内存共享模型实现进程间通信首先需要建立这样一块可以被相关进程读写的共享内存，这块内存事实上不在操作系统的控制之下。显然，为了数据的可靠性，必须要保证不能有同时超过一个的进程往这块共享内存中写数据。

信息传递模型可以有多种方式来实现，例如套接字、管道等。这些方式实现进程间通信的操作可以抽象为 send 和 receive。

2. 进程同步

前面提到利用共享内存模型实现进程间通信时必须保证不能同时有超过一个的进程往共享内存中写数据，事实上这也是进程同步的一个例子，这里要讲的进程同步主要通过所谓关键区域问题来展开。这里对关键区域做一下解释，假设同时有很多进程有一些共有的资源，而共有的资源不能同时被访问（有时候是修改），代码中需要访问共有资源的代码段称为关键区域。此外定义获取进入关键区域权限的代码段进入区域，释放进入关键区域权限的代码段称为退出区域，其余代码段称为剩余区域只。

要解决关键区域问题必须要满足下面三项要求：互斥（mutual exclusion），即若 p 进程在其关键区域中执行时其他相关进程都不能在其关键区域执行；前进（progress），即若没有进程在其关键区域中执行时则只有那些不在剩余区域中执行的进程可能被选择进入关键区域执行，并且这种选择不能无限推迟；有限等待（bounded waiting）：进程等待进入关键区域的时间必须有限。

解决关键区域问题可以有硬件或者软件方法，这些方法都基于所谓锁来实现的。先讲硬件实现的方法。硬件上可以实现一些禁止中断的指令或者说原子指令，就是这些指令开始执行的话就必须要一次性执行全部。课本上的 test_and_set()操作是原子操作（概念类似于原子指令，也通过硬件来实现），把公有的称为锁的变量传入这个操作，如果当前锁被占用，则申请进入关键区域的进程被阻塞，同时只有一个进程可以拥有锁。书上还举了另一个原子操作 compare_and_swap()原子操作，道理类似，不再详述。

有很多依靠软件的方法可以解决(并不一定是完美解决)关键区域问题，下面逐个叙述这些方法：

Peterson 方法：Peterson 方法适用于两个进程在关键区域与剩余区域交替执行。通过设置一个共有整型变量（书上用 turn）来表示哪一个进程可以进入临界区，当进程 Pi 希望进入关键区需把 turn 变量设为 i 并且进程 Pi 状态的变量指示为希望进入关键区域的状态即可。Peterson 方法满足前面叙述的三个要求。

互斥锁：使用互斥锁来保护关键区域防止数据冒险的方法是一个进程在进入关键区域之前必须要获得互斥锁，如果获得互斥锁的操作成功就进入关键区域否则就反复执行 while 循环获取锁直到成功（busy wating），因此这种互斥锁又被称为自旋锁。显然关键

区域执行结束之后占有互斥锁的进程需要释放互斥锁以便其他进程可以进入关键区域。需要注意的是这里获得互斥锁和释放互斥锁的操作都必须是原子的。

信号量：一开始的信号量实际上就是一个被进程共享的整型量，初始化为 1，书上开始实现的 wait 操作如果信号量 S<=0 的时候会进入忙等待，效率不高。后面修改了信号量的定义，定义了一个结构体为信号量，结构体中定义了一个整型量，和一个等待的进程队列，如果进程 wait 是整型量小于 0 就把进程加到等待队列中并阻塞，当有进程从关键区域中退出时从等待队列中弹出一个等待进程进入关键区域（signal 操作）。

死锁是进程同步中一个很重要的概念，机制设计不当或者程序员代码不正确都可能导致死锁（事实上现在有一些机制帮助粗心程序员解决这个问题）。死锁是指两个或多个进程无限期地等待其他的进程释放自己所需要的资源，由于可能都不释放资源就能导致无限等待，就形成了所谓死锁。

书上的几个经典进程同步问题，例如有限缓冲问题、读者-作者问题以及哲学家进餐问题都是利用信号量来解决进程同步的具体例子。篇幅有限不再详述。

3. 进程调度

进程调度分为长班调度和短班调度，这里说的是短班调度。调度用一句话来概括就是通过　某些策略来从等待的进程队列中选取进程放到 CPU 中执行。通常有几个标准来衡量一个调度的好坏程度：CPU 的利用程度（越高越好）、生产量（越多越好）、转向时间（越少越好）、等待时间（越少越好）以及反应时间（越短越好），这些概念都不必再详述。下面主要说明几种调度算法。

FIFO 算法：也叫 FCFS 算法，先进入 ready 队列的进程先执行，并且不允许抢占，显然这种方法不合理，也已经被现代操作系统淘汰了；

SJF 算法：在 ready 对列中选取所需在 CPU 执行时间最短的进程执行，这种算法同样被操作系统淘汰；

RR 算法：给每个进程分配一个初始的值，当进程的这个值为 0 时从 CPU 中释放并选择下一个进程轮换执行；

优先级调度和多级优先级调度以及多级反馈优先级调度：给每个进程根据一些标准赋予优先级，优先级高的进程先执行。若针对不同的优先级有不同的队列和调度策略这就是多路优先级调度，若不同级的队列中的进程可以相互调整进程则称为多级反馈优先级调度。

课本上还叙述了两种针对实时系统的调度算法，速率单调调度算法和截至期早先执行算法。这两种算法主要针对所谓强实时系统，即进入可执行进程队列的系统有一个执行完毕的截至时间，CPU 根据截至时间来确定是否可调度，若可调度则选择相应的算法调度使得所有的进程都能在截至时间之前完成，若无法完成调度则直接不执行无法在截至时间之前完成的进程。

进程调度的算法很多，虽然有一些能够评估调度算法优劣的度量，例如 CPU 利用率和相同时间工作量等，但不能就断言说某种调度算法最好也没有标准的调度算法，事实的情况往往是不同情况下应该选取不同的调度算法。

参考文献格式:

[1] Abraham Silberschatz. Operating System Concepts Essentials　Second Edition.Wilay 2014