进程管理

内核的一个主要功能，在内核态完成

进程的概念和特征

进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。

当前一操作（程序段）执行完成后才能执行后续操作，执行期间要从前往后顺序执行。但一旦引入并发程序后，程序将有间断性、失去封闭性、不可再现性。

进程的特征

1. **结构特征**：程序段、数据段、PCB 三个部分组成进程实体
    - 进程控制块PCB：OS根据PCB来对并发挥自i选哪个的进程进行控制和管理。其所包含的信息有：
        - 进程标识符：外部标识符PID，内部标识符(端口)
        - 处理机状态：通用寄存器，指令计数器，程序状态字PSW，用户栈指针
        - 进程调度信息：进程状态，进程优先级，进程调度所需的其它信息，事件
        - 进程控制信息：程序和数据的地址，资源清单，进程同步通信机制，链接指针
2. **动态性**：进程的最基本特征，由创建而产生、由调度而执行、由撤销而消亡。
3. **并发性**：多个进程实体同存于内存，且能在一段时间内同时运行。*并发*是多个进程轮替执行，*并行*是同时执行。
4. **独立性**：进程实体是一个能独立运行、独立分配资源、独立接受调度的基本单位。但凡为建立PCB的程序都不能作为一个独立单位参与运行。
5. **异步性**进程按各自独立的、不可预知的速度向前推进（进程实体按异步方式运行）。

进程控制块的组织方式

1. 链接方式：把进程按照链表的形式连接起来。
2. 索引方式：用索引表来存储进程控制块的位置信息。

进程和程序的关系

进程是程序在数据集上的一次执行；程序是构成进程的组成部分，一个程序可对应多个进程，一个进程可包含多个程序；集成度运行目标是还行所对应的程序；从静态看，进程由程序、数据、进程控制块PCB组成。

进程	**程序**
动态	静态
是程序在数据集上的依次执行，有创建、撤掉，存在是暂时的	是指令的有序集合，永远存在
有并发性	无并发性
进程可创建其它进程	程序不能形成新的程序
竞争计算机资源的基本单位	不参竞争计算机资源

进程的状态和切换

进程的五种基本状态

- 创建状态：正在创建，不能运行，未提交，等待系统完成创建进程的所有必要信息，并初始化PCB
- 就绪状态：具备运行条件，等待分配CPU
- 执行状态：占用CPU并运行
- 阻塞状态：因某件事发生导致暂时无法运行（例如等待IO操作）
- 终止状态：执行完最后语句，并通过系统调用exit()请求OS终止进程，并回收内存空间和PCB

五种状态的转换

- 创建：创建进程->
- 就绪：调度进程->
- 运行：->
    - 终止进程->终止
    - 等待IO操作/Buffer/执行P操作/申请内存/申请设备->阻塞
        - 等待事件完成->就绪
    - 时间片用完/CPU被剥夺/CPU被高优先进程抢占->就绪

进程控制

对系统中的所有进程实施有效管理，具有创建新进程、撤销已有进程、实现进程状态转换等功能。进程控制具有原子性，属于原语结构。
进程之间的组织关系是进程图。

1. 进程创建：进程创建原语，引起创建进程事件、进程的创建过程、创建完成
2. 进程终止：检索被终止进程的PCB，立即终止进程、调度标志置为真，子孙进程终止，进程拥有资源归还父进程或系统，被终止进程的PCB从所在队列或链表中移出
    1. 正常结束：一个用于表示进程已经运行完成的指示。
    2. 外界干预（进程相应外界请求而终止运行）：操作员或操作系统敢于、父进程请求、父进程终止
    3. 异常结束：出现错误或故障被迫进程终止
3. 进程阻塞：进程无法继续执行时调用阻塞原语block把自己阻塞，进程控制块由“执行”改为阻塞，PCB插入阻塞队列，调转读程序进行重新调度
    - 原因：请求系统服务，启动某种操作，新数据尚未到达，无新工作可做
4. 进程唤醒：被阻塞进程所期待事件出现，有关进程调用唤醒原语wakeup()
5. 进程挂起：进程被暂时调离出内存，条件允许时被操作系统再次调回内存，重新进入就绪状态。被挂起的程序由就绪改为静止就绪；阻塞改为禁止阻塞；正在被执行则专项调度程序重新调度。
    - 原因：用户请求，父进程请求，操作系统要求，负载调节要求，定时任务
6. 进程激活：父进程或用户进程请求激活指定进程，且内存已有足够空间，OS利用激活原语active()激活指定程序。静止就绪改为活动就绪；静止阻塞改为活动阻塞
7. 进程调度：

进程控制的原语

1. 进程创建原语：申请一个空白PCB，将调用者参数添入PCB，设置记账数据，设置新进程为“就绪”
2. 终止进程原语：终止已完成的进程，回收其所占资源
3. 阻塞原语：将进程从运行态变为阻塞态。进程被插入等待事件队列，同时修改PCB中相应表项
4. 唤醒原语：将进程从阻塞态变为就绪态。进程从阻塞队列移出，插入就绪队列等待调度，同时修改PCB相应表项

进程同步

两个及以上进程基于某个条件协调活动。

进程具有异步性。但进程与进程需要共享系统资源，将构成竞争关系，于是存在**制约关系**：

- 间接相互制约关系：互斥——竞争
- 直接相互制约关系：同步——协作

资源竞争会出现两个控制问题： **死锁deadlock**（一组进程获得了部分资源并想得到其它进程占有的资源，最终所有进程进入死锁）；
饥饿starvation（由于其它进程优先于它而被无限延期）

进程的互斥是解决进程间竞争关系的手段（若干个进程要使用同一资源时，最多允许一个进程去使用，其它进程等待。此过程要借助临界区）

临界区是进程访问临界资源的代码段（临界段）：进入区、临界区、退出区、剩余区。进入区和退出区是负责实现互斥的代码段。

进程同步的原则

1. 空闲让进：临界资源处于空闲状态，允许一个请求进入临界区的进程立即进入自己的临界区，以有效利用临界资源。
2. 忙则等待：临界区已有进程时，其它进程必须等待，保证临界资源的互斥访问。
3. 有限等待：保证进程在有限时间内能进入临界区，避免陷入死等
4. 让权等待：当进程不能进入临界区，应立即释放出立即，避免陷入忙等

1、2、3、必须满足，4、可以不被满足

进程同步的软件实现

单标志法、爽标志先检查、双标志后检查、Petersor算法

单标志法：
设置一个公共整型变量turn，代表是否允许进入临界区的进程编号。
但若某一进程不想进入临界区且turn==True，则其它进程也无法进入临界区，出现临界区资源空闲状态，违背*空闲让进*原则。

双标志先检查：
设置一个flag表示自己的访问状态，进程访问临界区前检查临界资源是否被占用，并设置自己的flag。这样不用交替进入，可以连续使用。
但两个进程会容易都进入临界区，违背*忙则等待*原则。

双标之后检查：
进程访问前先设置自己的flag，再检查其它进程的flag。两个进程都会表明自己要进入临界区的意愿。
但容易让两个进程都处于等待状态，不满足*有限等待*，导致饥饿。

Peterosn算法：
结合单标志法和双标志后检查，利用flag解决互斥访问，利用turn解决饥饿。
防止进程为了进入临界区而出现无限等待，再设置变量turn。进程设置flag后再设置turn标志。

在while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。
但不遵循*让权等待*，会发生忙等。

进程同步的硬件实现方法

关中断，利用Test-and-Set指令实现互斥，利用swap指令实现进程互斥

关中断屏蔽方法：
利用开关中断指令实现。整个指令执行过程不允许进程切换，不可能存在两个进程同时都在临界区。简单高效。
但不适合多处理机，只适合操作系统内核进程；让用户随意使用会影响整个CPU调度。

TestAndSet指令：
称TSL指令，配合互斥锁，确保只有一个进程在临界区。如果锁可用，进程继续执行并将锁设置为占用状态，其它进程会被阻塞，直到锁可用。
实现简单，不用严格检查逻辑漏洞，适用于多处理机。
但不满足让权等待，暂时无法进入临界区的进程会占用CPU并循环执行，循环指令TSL指令，导致忙等。

swap指令：
逻辑上和TSL指令无太大区别。临界区被占用，lock为true，交换后依然进入while循环；临界区空闲，old为false，跳出循环，进程进入临界区。
优缺点和TSL类似。

进程同步机制（信号量机制）

整型信号量、记录型信号量。

用户可以用操作系统的一对原语对信号量进行操作，便于实现进程同步、互斥。信号量是一个可以表示系统中某种资源数量的变量。

整型信号量

由Dijkstra把整型信号量定义为一个整型量，除初始化外，金童两个标准原子操作wait(S)和signal(S)访问（分别称为P、V操作）。

```
wait(S): while S≤0 do no-op
   S:=S-1
signal(S): S:=S+1
```

只要信号量S≤0，则会不断地测试。因此不遵守让权等待，使进程处于忙等。

记录型信号量

需要用于代表资源数目的整型变量value、进程链表L（链接所有等待进程）。

```
typedef struct{
   int value;  //剩余资源数
```

```
    struct process *L;    //等待资源队列
}semaphore;
```

```
procedure wait(S)
   var S: semaphore;
   begin
      S.value: = S.value-1;
      if S.value < 0 then block(S,L)
   end
procedure signal(S)
   var S: semaphore;
   begin
      S.value: =S.value+1;
      if S.value ≤ 0 then wakeup(S,L);
   end
```

S.value的初始值取决于系统汇总有多少可用资源数量。
当请求一个单位的资源时：`S.value: =S.value-1`，当S.value<0时，表示资源分配完毕，调用block自我阻塞，并插入到信号量链表S.L，遵循让权等待原则。
如果S.value初值为1，表示只允许一个进程访问临界资源，此时信号量转为互斥信号量。 signal操作则表示执行进程释放一个单位资源，若S.value+1后仍是S.value≤0，表示信号量链表中仍有等待该资源的集成被阻塞，古调用wakeup将表中的第一个等待进程唤醒。

管程机制

管程是代表共享资源的数据结构、以及由对该共享数据结构实时操作的一组过程锁组成的资源管理程序，共同构成了一个操作系统的资源管理模块。

由四部分组成：管程名称，局部于管程内部的共享数据结构说明，对该数据结构进行操作的一组过程，对局部于管程内部的共享数据设置初始值的语句。

管程的特性：模块化，抽象数据类型，信息掩蔽。

进程和管程对比：进程通过调用管程中的过程对共享数据结构进行实时操作，管程不能像进程那样并发执行（不能与其调用者并发），管程是操作系统中的一个供进程调用的资源管理模块（不具有动态性）。

管程相当于围墙。局部于管程内部的数据结构仅能被局部于管程内部的过程所访问。管程外与管程内的无法相互访问。所有进程要访问临界资源时，必须经过管程，且每次只允许一个进程进入管程，实现进程互斥。

一个进程被阻塞或挂起的条件/原因可能很多，因此管程中设置多个**条件变量condition**，每个条件变量保存一个链表用于纪录因该条件变量而阻塞的所有进程，**用x.wait、x.signal操作。**
当进程因x条件被阻塞/挂起，x.wait把此进程放到一个因x原因引起而阻塞的队列上。
当x条件变化，x.signal让x阻塞队列上的一个进程重新启动一个被阻塞的进程（如果没有被阻塞的过程，此操作不产生任何后果）。此signal不同于信号量机制的signal。

锁机制

锁让某一时间点只有一个线程进入临界区代码，保证数据一致性。锁的本质是内存中一个用于表示不同状态的整型数。
加锁时，判断锁是否空间，若空闲则修改为加锁状态并返回成功，若已上锁则返回失败，解锁时则修改为空闲状态。

锁的分类

1. **重量级锁**：获得不了锁时立即进入阻塞。
2. **自适应自旋锁**：根据线程最近获取锁的状态调整循环次数的自旋锁，称为自适应自旋锁。
3. **轻量级锁**：引入一个变量标记使用情况。若未在使用，则当进入此方法时采用CAS机制把状态编辑为已有人在执行，退出时改为无人执行。
4. **偏向锁**：偏向锁认为很少有两个线程执行一个方法，于是没必要加锁。
5. **悲观锁和乐观锁**：
    - 悲观锁认为不事先加锁总会出事。重量级锁、自旋锁、自适应自旋锁属于悲观锁。
    - 乐观锁认为不加锁没事，出现冲突了再解决（例如CAS机制）。轻量级锁属于乐观锁。

互斥锁

1. **LockOne类**：引入一个标志数组flag用于标记进入临界区的意愿。但当两个线程都表达想进入临界区的意愿，则会死锁。
2. **LockTwo类**：当两个进程都表达想进入临界区的意愿时，牺牲较晚对victim赋值的线程，以避免死锁。但只有一个进程时，会牺牲自己。
3. **Peterson锁**：LockOne和LockTwo的结合。
4. **Barkey锁**：n线程锁算法。进程进入临界区前flag表达意愿，并得到一个序号，序号最小的线程才能进入临界区。

进程通信

进程之间的信息交换。进程之间存在相互制约的关系（间接，直接）。

进程间通信机制IPC：进程之间交换数据必须通过内核，把进程1的数据从用户控件拷贝到内核缓冲区，进程2再从缓冲区读取。

低级通信：信号量机制，管程。
此通信方式效率低（生产者每次只能向缓冲池投放一个消息，消费者每次只能从缓冲区获取一个消息），通信对用户不透明

高级通信方式：共享存储器系统，消息传递系统，管道通信。

共享内存

多个进程直接读取同一块内存空间，最快的可用IPC形式。在内核专门留出一块内存区，由需要访问的进程将其映射到自己的私有地址空间供进程直接读写。

由于多个进程共享一段内存，依靠某种同步机制（如信号量）实现进程间的同步和互斥。

基于数据结构共享：共享空间内只放一个数组。速度慢、限制多，一种低级通信方式。

基于存储区的共享：OS在内存画出一块共享存储区，数据的形式、存放位置由通信进程控制。速度快，一种高级通信方式。

消息Message队列

在消息传递系统中进行进程间的数据交换，以格式化的消息为单位（计算机网络中把message称为报文）。程序员直接利用系统的提供的一组通信命令（原语）进行通信。OS隐藏通信实现细节，简化通信程序编制的复杂性。
分为直接通信和间接通信。

直接通信方式：发送进程利用OS的发送命令直接把消息发送给目标进程（`Send(Recevier,message); Receive(Sender, message);`）。

间接通信方式：以“信箱”为中间实体进行消传递。

管道pipe

连接一个度进程和一个写进程以实现通信的共享文件（不隶属于文件系统）。

管道只能采取半双工通信（某一时间段内只能实现单向传输），个进程互斥访问管道（由OS实现），任意时刻只有一个进程对管道进行操作。

线程与进程

进程是系统资源分配的单位，线程是处理器调度的单位。

为了让程序能并发执行，系统必须进行：创建进程，撤销进程，进程切换。
进程时进程实体的运行过程，是系统进行资源分配与调度的一个独立单位。将这两个属性由操作系统分开处理，并对拥有资源的基本代为不进行频繁切换，于是形成了线程的概念。

-	**进程**	**线程**
组成	程序段、数据、PCB	共享其隶属的进程，但拥有自己的线程ID、寄存器等
并发性	没有引入线程的系统重，进程是独立运行的基本单位	线程是独立的基本单位，一个进程至少拥有一个线程
资源	进程是资源分配和拥有的基本单位	共享其隶属于进程的资源，但也拥有自己的线程资源
调度	没有引入线程的系统中，进程是独立调度的基本单位	在引入线程的系统重，线程是独立调度的基本单位
通信	信号量，共享存储，消息传递系统，管道通信	同一个进程内的线程直接共享该进程的数据段，不同进程属于进程通信
地址空间	相互独立	同一个进程的各个线程共享该进程地址空间

线程的属性

1. 轻型实体：线程基本不拥有系统资源（只保留必不可少能保证独立运行的资源
2. 独立调度和分派的基本单位（由于是轻型实体，所以调度速度快）
3. 可并发执行
4. 共享进程资源

线程的状态参数（线程独有的且无法共享的）：寄存器状态，堆栈，线程运行状态，优先级，现成转悠存储器，信号屏蔽。

多线程OS中的进程

进程作为拥有系统资源的基本单位，包含多个线程并提供资源（但此时进程不再作为一个执行实体）

多线程OS中的进程包含的属性：作为系统资源分配的单位，可包含多个线程，进程不是一个可执行实体。

线程的实现

用户级线程、内核级线程

用户级线程

进存在于用户控件，创建、撤销、线程间同步与通信等，都无需OS调用。

可以在不支持线程操作系统中实现，创建、销毁、切换等线程管理代价比内核线程少，允许每个进程定制调度算法，管理灵活，能够利用的表空间和堆栈空间比内核级线程多。
但同一进程只能有一个线程同时运行，且如果线程使用系统调用使其被阻塞，整个进程都会挂起。页面失效也会产生类似问题。

内核级线程

在内核支持下运行，创建、撤销、切换都依靠内核实现。为每一个内核支持线程设置了一个线程控制块，内核根据该控制块而感知线程存在，并加以控制。

某一线程被阻塞，其它线程依旧可以并发运行，不同的线程可以进入不同的处理机运行。
但进程切换慢，线程管理开销大。

多模型线程

1. 多对一模型：不需要切换到核心态，系统开销小，效率高；但当一个线程被阻塞后，整个进程被阻塞。
2. 一对一模型：一个线程被阻塞后其它线程依旧继续运行，并发能力强；但一个用户进程会占用多个内核线程，切换到内核态开销大，线程管理成本高。
3. 多对多模型：多对一和一对一的折中。

补充内容

- 设备分配依据设备分配表，不需要创建新进程
- 进程的阻塞或唤醒不是撤销与建立的过程
- 进程自身决定从运行状态到阻塞状态
- P操作(wait(S))可能会导致进程阻塞
- P、V操作实现进程同步，信号量初值由用户决定
- 同一进程或不同进程内的线程都可以并发执行