1.引入进程的原因：

在多道程序环境下，允许多个程序并发执行。当多个程序同时运行时，各程序会共享系统资源，各程序在执行的时候有着相互制约的关系。因此，各程序失去了**封闭性**，并具有**间断性**以及**不可再现性**，而这些特征都是程序在执行的过程中生成的，是动态的；而程序是一系列指令的集合，是一静态概念。因此，“程序”就无法描述程序的执行在内存中的情况。引入进程，以便能更好的描述和管理程序的执行，支持和管理多道程序环境中的并发执行，实现操作系统的并发性和共享性（两大基本特性）。

串行执行：



并行执行：



2.进程的概念：

① 进程是进程实体的运行过程，是操作系统进行**系统资源**调度和分配的基本单位。

【（此处的系统资源不是“死的”，是指处理机、存储器和其他设备 服务于某个进程的“时间”。例如“处理机资源”就是指处理机的时间片。）（进程是系统资源和调度的基本单位，即这些“时间片”是分配的独立单位，这就决定了进程是一个动态、过程性的概念）】

② 进程是程序的一次执行过程。

③ 进程是一个程序及数据集在处理机上**顺序执行**时所发生的活动。

④ 进程是具有独立功能的程序在数据集合上的运行的过程，是系统进行资源调度和分配的一个独立单位。

3.进程的特征：

① 动态性：

进程是程序的一次执行过程。动态的产生、变化和消亡的。【最基本特征】

② 并发性：

内存中同时会有多个进程实体，各进程可以并发的执行。

③ 独立性：

进程是独立运行的、独立获取资源和独立接受调度的基本单位。

④ 异步性：

由于各个进程的相互制约，使得各个进程具有执行的间断性，即各进程按各自独立的、不可预知的速度向前推进。异步性会使得执行结果的不可再现。（对这个问题，通过进程同步机制来解决）

⑤ 结构性：

每个进程都配置一个PCB。从结构上，进程实体都由PCB、程序段、数据段组成。

4.进程的组织：

① 进程控制块(PCB, Process Control Block)：

【（为了使参与并发执行的程序（含数据）**能够独立地运行**，必须为此配置一个专门的数据结构，即PCB。系统通过PCB来描述进程的基本信息和运行状态等，进而控制和管理进程。）】

【PCB是进程存在的唯一标志。进程创建时会创建相应的PCB，进程终止时会撤销相应的PCB。】

* 进程描述信息： （可用于区分进程）

进程标识符（PID）：每个进程有唯一的标识符。

用户标识符（UID）：主要为共享和保护服务。

* 进程控制和管理信息：

进程的当前状态（用State变量表示，就绪态？？运行态？？阻塞态？？）。

进程的优先级：描述进程抢占处理机的优先级。

代码运行入口地址

程序的外存地址

进入内存的时间

处理机占用的时间

信号量使用

* 资源分配清单： （可实现操作系统对资源的管理）

代码段指针

数据段指针

堆栈段指针

文件描述符

键盘

鼠标

* 处理机相关信息： （可实现操作系统对进程的管理和调度）

地址寄存器值

控制寄存器值

标志寄存器值

通用寄存器值

状态字

【在一个系统中，通常存在着许多处于不同状态的进程。为了方便进程的管理和调度，需将同状态的各进程的PCB组织起来。组织方式：① 链式方式 ② 索引方式】【链式方式：OS拥有指向各队列的指针；索引方式：OS拥有指向各索引表的指针】

② 程序段： 被进程调度程序调度到CPU执行的程序代码段。

③ 数据段：可以是对应程序加工处理的原始数据，也可以是程序执行时产生的中间值或最终结果。

进程实体：一个进程实体由PCB、程序段、数据段组成。进程是动态的，进程实体是静态的。

5.进程的状态及状态转换：

① 创建态（新建态）：

进程正在被创建时所处的状态。系统分配所需资源，初始化PCB。

【若在创建时，进程所需的资源尚得不到满足，比如系统无足够的内存将进程装入其中，此时处于创建态。**OS可以根据系统性能或主存容量的限制推迟新进程的提交**（创建状态）。对于处于创建态的进程，当其获得了所需的资源以及对PCB初始化工作的完成后，便可从创建态转入到就绪态。】

② 就绪态：

进程创建完成后处于就绪态。处于就绪态的进程已经具备运行条件，但由于此时没有空闲的处理机，就暂时不能运行。

③ 运行态（执行态）：

进程正在CPU上被执行时所处的状态。单核计算机中，任何时刻最多只有一个进程处于运行态。

④ 终止态（结束态）：

系统可能是正常终止或异常终止。此时进程处于终止态，操作系统让该进程下CPU，并且回收内存空间等资源，最后撤销PCB。

【进程进入终止态后不能再执行，但在OS中依然保留一个记录，其中保存状态码和一些计时统计数据，供其他进程收集，一旦其他进程完成了对其信息的提取之后，OS将删除该进程，即将其PCB清零，并将空白的PCB返还给系统。】

⑤ 阻塞态（等待态）：

进程在运行的过程中可能会**主动**的请求等待某个事件的发生（如等待某系统资源的分配，或等待其他进程的响应），在这个事件发生之前，进程无法继续往下执行，操作系统会让该进程下CPU，并使其处于阻塞态。

【挂起状态：当内存空间不足、系统承载过大时，操作系统会让处于暂时不能运行的进程放到外存挂起。然后，当内存有空闲时，操作系统通过中级调度（内存调度）将处于挂起状态的进程激活。挂起操作和激活操作是通过**挂起原语**Suspend、**激活原语**Active实现的。】

⑥ 就绪挂起：将处于就绪态的进程挂起。

⑦ 阻塞挂起：将处于阻塞态的进程挂起。

【未挂起的就绪态称活动就绪态(Readya），挂起的就绪态称静止就绪态（Readys）；未挂起的阻塞态称活动阻塞态(Blockeda），挂起的阻塞态称静止阻塞态（Blockeds）；】

状态转换：

创建态🡪就绪态🡪运行态🡪终止态

运行态🡪阻塞态🡪就绪态

运行态🡪就绪态

就绪态🡨🡪就绪挂起（静止就绪态）

阻塞态🡨🡪阻塞挂起（静止阻塞态）

阻塞挂起🡪就绪挂起（静止就绪态）

创建态🡪就绪挂起（静止就绪态）

运行态🡪就绪挂起（静止就绪态）



【[就绪态是指进程除了CPU资源以外已经获得了其他所需的资源，而等待态是指进程除了CPU资源以外还要等待其他时间发生（如某系统资源的分配，或其他事件的响应），那么为何要将CPU资源和其他资源分开？？？] [因为在**分时系统**的**时间片轮转机制**中，每个进程分到的进程片是若干毫秒，即进程得到处理机的时间很短且非常频繁，进程在运行过程中实际频繁转换到就绪态。而其他资源（如外设）的使用和分配或某一事件的发生（如I/O操作的完成）对应的时间相对来说很长，并且进程转换到等待态的次数也相对较少。]】

6.进程的控制：

进程控制的主要功能是对系统中的进程实施有效的管理，它具有创建新进程、撤销已有进程、实现进程的状态转换等功能。

进程的控制是通过**原语**来实现的。

【进程控制的不可中断性：进程的控制是“一气呵成”的，否则会使操作系统中某些关键的数据结构信息的不统一，会影响操作系统的其他管理功能。】（如， 若此时有进程要从阻塞态转换到就绪态，那么操作系统中负责进程控制的内核程序至少要执行两个步骤（当然还包括其他的一些如进程上下文的保存等）：a.将该进程的PCB中表示进程状态的State变量进行更改 b.将该PCB从阻塞队列插入到就绪队列；若此时在执行完步骤a后，发生中断，则该进程被暂停执行，执行处理相应中断的处理程序；那么，此时系统中这个PCB对应的进程是处于就绪态，而这个PCB是在阻塞队列中。）

【原语：是OS内核中一种特殊的程序，具有原子性。】

【原语的原子性实现：是通过**关、开中断指令**两个特权指令实现的。】（原语是在OS内核中，需要在“内核态”下运行。一般CPU在执行完一条指令后，会例行检查此时是否有需要处理的中断信号，若有，则停止执行该程序，转而去执行处理相应中断的处理程序；而若当执行了关中断指令后，CPU在执行完指令之后不再去检查是否有需要处理的中断信号，直至执行了开中断指令后，恢复之前的操作。）

1. 进程的创建：

【进程可以创建进程，创建的称父进程，被创建的称子进程；进程之间的关系是树形结构。子进程可以继承父进程所拥有的的资源。当子进程被撤销时，应将从父进程那里获得的资源归还给父进程。】

创建原语：

* 为进程分配一个唯一的PID，并申请一个空白的PCB（PCB是有限的）。若PCB申请失败，则创建失败。
* 为进程分配资源，为新进程的程序、数据和用户栈分配必要的内存空间。（若此时资源不足（如内存空间），则并不是创建失败，而是处于**创建态**。）
* 初始化PCB。
* 若此时进程就绪队列能够接纳新进程，则将新进程插入就绪队列，等待被调度。

引发进程创建的事件：

终端用户登录、作业调度、系统提供服务、用户进程的应用请求（用户进程自己创建进程）。

1. 进程的终止：

撤消原语：

* 根据被终止进程的PID，检索PCB，从中读出该进程所处的状态。
* 若被终止进程处于运行态，则立即终止该程序的执行，将PCB分配给其他进程。
* 若该进程还有子进程，则应该终止所有的子进程。
* 将该进程拥有的所有资源归还给父进程或操作系统。
* 将PCB从相应的队列中移除。

引发进程终止的事件：

* 正常终止： （如通过exit系统调用终止）
* 异常终止：

（进程在执行的过程中遇到不可修复的错误，如存储区越界、非法指令、特权指令错、I/O故障、保护错等）

* 外界干预：（如用户强行杀死进程）

1. 进程的阻塞与唤醒：

阻塞原语Block： 【是由被阻塞进程**自我调用**实现的】

* + - 找到要阻塞的进程对应的PCB
    - 保护进程运行的现场，将PCB的状态信息置为“阻塞态”，暂时停止运行
    - 将PCB插入相应事件的等待队列中

引发进程阻塞的事件：

* 需要等待系统分配某种系统资源
* 需要等待相互合作的进程完成工作

唤醒原语Wakeup： 【是由一个与被唤醒进程**合作**或被其他相关进程**调用**实现的】

* 在该事件的等待队列中找到PCB
* 将PCB从等待队列移出，设置进程状态为“就绪态”
* 将PCB插入就绪队列，等待被调度程序调度

引发进程唤醒的事件：

等待事件的发生

1. 进程的切换：

切换原语：

* 保存处理机上下文
* 更新PCB信息
* 把进城的PCB移入相应的队列
* 选择另一个进程执行，并更新其PCB
* 更新内存管理数据结构
* 恢复处理机上下文

引发进程切换的事件：

* 当前进程的时间片到
* 有更高优先级的进程到达
* 当前进程主动阻塞
* 当前进程终止

【**进程切换与处理机模式切换**是不同的。模式切换时，处理机**逻辑上可能还在同一个进程中运行**。若因中断或异常进入核心态运行，执行完后又回到用户态刚被中断的程序，则操作系统**只需恢复进程进入内核时所保留的处理机现场**，而**无需改变当前进程的运行环境**；但是若要切换进程，当前运行进程该变了，则当前进程的运行环境也需要改变。】

【“调度”和“切换”的区别：调度是决定资源分配给哪个进程的行为，是一种决策行为；切换是指实际分配的行为，是一种执行行为。】

7.进程的通信：

进程的通信即进程间的信息交换。

进程是分配资源的基本单位（包括内存地址空间），因此各个进程拥有的内存地址空间是相互独立的。因此，一个进程不能直接访问另一个进程的内存地址空间。

1. 共享存储：

【操作系统为进程分配一个**可直接访问的共享空间**，通过这个共享空间的**读/写操作**来实现进程间的信息交换。】

【各个进程对共享空间的访问必须是互斥的（互斥访问是通过操作系统提供工具实现）】

【操作系统只负责为通信进程**提供可共享使用的存储空间以及同步互斥工具**（如P操作、V操作），而进程间的数据交换则是由**用户自己安排读/写指令完成**。】

* 基于数据结构的共享：【速度慢、限制多，是一种低级通信方式】

（如共享空间只能存放一个长度为10的数组。）

* 基于存储区的共享：【相比基于数据结构共享，速度更快，是一种高级的通信方式】

内存中数据的形式、存放位置是由通信进程控制的，而不是由操作系统。

1. 消息传递：

【若通信进程间不存在可直接访问的共享空间，则需通过利用系统提供的消息传递的方法实现进程通信。】

【在消息传递系统中，进程间的信息交换是以**格式化的消息**（Message）为单位的。进程通过操作系统提供的“**发送消息**/**接收消息**”两个原语来进行数据交换的。】

【格式化的消息：由消息头（包括进程PID、接收进程PID、消息类型、消息长度等格式化信息）和消息体组成】

* 直接通信方式：

（首先由发送进程创建格式化的消息，通过发送消息原语将格式的信息发送，并将它挂在接收进程的消息缓冲队列上，然后接收进程通过接收消息原语依次读取其消息缓冲队列。）

* 间接通信方式（信箱通信方式）：

（首先由发送进程创建格式化的消息，通过发送消息原语将信息发送到中间体（信箱），然后接收进程通过接收消息原语从中间体取得格式化的消息。）

1. 管道通信：

【“管道”：用于连接读/写进程的以实现它们之间通信的一个共享文件（pipe文件）。其实就是在内存中开辟了一个大小固定的缓冲区。（Linux中，一个管道对应缓冲区的大小是4KB）】

【管道只能采用半双工通信，某一时间内只能实现单向的传输；若要实现双向同时通信，则需要设置两个通道。】

【各个进程需要互斥地访问管道；】

【数据以字符流的形式写入管道，当管道写满时，写进程的write()系统调用将被阻塞，等待读进程将数据取走；当读进程将数据全部取走后，管道为空，读进程的read()系统调用将被阻塞；】

【如果管道没写满，则不允许读；若管道没读完，则不允许写】

【从管道读数据是一次性操作，即数据一旦从管道读出，就要被管道抛弃，这意味着读进程最多只有一个，否则可能会出现错读数据的情况；】

8.线程：

【进程的时间代价：进行进程切换、同步及通信等所付出的时间开销。】

【进程的空间代价：进程控制块及协调各运行机构所占的内存空间开销。】

引入目的：

为了减小程序在并发执行时所付出的时空开销，提高操作系统的并发性能。

传统的进程是程序执行流的最小单位；而引入线程后，线程成为程序执行流的最小单位。

线程就是“轻量级的进程”，是一个基本的CPU执行单元，也是程序执行流的最小单位。由TID、TCB组成。

引入线程之后，不仅进程之间可以并发，进程内的各线程之间也可以并发，从而提高了系统的并发度，使得在一个进程内可以处理各种任务。

引入线程之后，进程只作为除CPU之外的系统资源的分配单位。

进程与线程比较：

* 调度：

传统操作系统中，进程是拥有资源、调度的基本单位；

引入线程机制后，进程是拥有资源的基本单位，线程是调度的基本单位；

* 拥有资源：

在线程引入前及引入后，进程都是拥有资源的基本单位；而线程不拥有系统资源（也有一点必不可少的资源），但线程可以访问隶属进程的系统资源。

* 并发性：

传统操作系统中，只能进程间并发；

引入线程机制后，不仅进程间可并发执行，多个线程间也可并发执行；

* 系统开销：

由于创建和终止进程时，操作系统都要为之分配或回收资源，如I/O设备、内存空间等，因此操作系统所付出的开销远大于创建和撤销线程时的开销；

在进行进程切换时，涉及到当前执行进程的CPU环境保存及新被调度的进程的CPU环境设置，而线程切换只需保存和设置少量地寄存器内容，开销很小；

由于一个进程内的多个线程共享进程的资源，因此，某进程内的线程间的同步和通信易于实现，甚至无需操作系统的干预；

* 地址空间和其他资源：

进程的地址空间相互独立，同一进程内的线程共享进程的资源，某进程中的线程对另一进程是不可见的。

* 通信方面：

进程间的通信（IPC）需要进程同步和互斥手段的辅助，以保证数据的一致性；而线程间可直接通过读/写进程数据来进行通信。

线程的属性：

线程是处理机调度的单位；

多核CPU计算机中，各个线程可占用不同的CPU；

每个线程都有一个线程ID和线程控制块TCB；

线程也有就绪、阻塞、运行三种基本状态；

线程几乎不拥有系统资源；

同一进程的不同线程共享进程的资源；

由于共享内存空间地址，同一进程的各个线程间通信甚至无需系统干预；

同一进程的线程切换，不会引起进程的切换；

不同进程的线程切换，会引起进程的切换；

切换同一进程内的线程，系统开销小；

切换不同进程内的线程，系统开销大；

线程的实现方式：

* 用户级线程：（User-Level Thread，ULT）

【早期的操作系统只支持进程，不支持线程，当时的“线程”是通由线程库来实现的。很多的编程语言提供了线程库，可以实现线程的创建、销毁、和调度等功能。】

① 用户级线程是由应用程序通过线程库来实现的。所有的线程管理工作都由**应用程序负责**（包括线程的切换）

② 用户级线程下，线程的切换是在**用户态**下即可完成，无需操作系统干预

③ 对于操作系统，并不能意识到线程的存在

优点：用户级线程的切换在用户态下即可完成，不需切换到核心态，线程管理的系统开销小、效率高

缺点：当一个用户级线程被阻塞，整个进程都会被阻塞，并发度不高。多个线程不能在多核CPU上并行运行。

* 内核级线程：(Kernel-Level Thread，KLT) 【操作系统支持的线程】

① 内核级线程的管理工作是由操作系统内核完成的；

② 线程的调度和切换等工作都是由内核负责，因此内核级线程的切换必须在**核心态**下才能完成；

③ 操作系统会为每个内核级线程建立对应的TCB。通过TCB对线程进行管理。

多线程模型：（在支持内核级线程的系统中，根据ULT和KLT映射关系，有以下三种划分）

* 一对一模型：

一个ULT映射到一个KLT。每个用户进程拥有与ULT相同数量的KLT。

优点：当一个线程被阻塞后，其他的线程还可以继续执行，并发能力强，多线程可以在多核CPU上并行执行

缺点：每创建一个用户级线程都要创建一个内核级线程与其对应，这样创建线程开销大，会影响到用户程序的性能

* 多对一模型：

多个ULT映射到一个KLT。且一个进程只能分配一个KLT。

优点：用户级线程切换在用户态即可完成，不需要切换到核心态，因此线程管理的系统开销小、效率高

缺点：当一个ULT被阻塞，整个进程被阻塞，并发度不高。多个用户级线程不能在多核CPU上并行执行

* 多对多模型：

n个ULT映射到m个KLT（n>=m）。每个用户进程对应m个KLT。

优点：克服了多对一模型并发度不高的缺点，又克服了一对一模型中一个用户进程占用太多的内核级进程而系统开销太大的缺点。

【用户级线程可以理解为“代码逻辑”的载体。内核级线程可以理解为“运行机会”的载体。内核级线程才是处理机分配的单位。一段“代码逻辑”只有在获得“运行机会”才能被CPU执行】

【内核级线程中可以运行任意一个有映射关系的用户级线程，只有当这个进程的所有内核级线程被阻塞，这个进程才会被阻塞】



9.进程上下文、CPU上下文、中断上下文

CPU总处于以下三种状态：

1. 内核态，运行于**进程上下文**，内核代表**进程运行于内核空间**；
2. 内核态，运行于**中断上下文**，内核代表**硬件运行于内核空间**；
3. 用户态，运行于用户空间；

CPU上下文：

CPU的状态信息,主要是由**CPU的各种寄存器中的内容**构成。

进程上下文：

Linux中，把系统提供给**进程的处于动态变化的运行环境总和**称为进程的上下文。

在Linux中，用户程序装入系统形成一个进程的实质是系统为用户程序提供一个完整的运行环境。进程的运行环境是由它的程序代码和程序运行所需要的数据结构以及硬件环境组成的。进程的运行环境主要包括：

* + 1. 进程空间中的代码和数据、各种数据结构、进程堆栈和共享内存区等。
    2. 环境变量：提供进程运行所需的环境信息。
    3. 系统数据：进程空间中的对进程进行管理和控制所需的信息，包括进程任务结构体以及内核堆栈等。
    4. 进程访问设备或者文件时的权限。
    5. 各种硬件寄存器。
    6. 地址转换信息。

中断上下文：

硬件通过触发信号，导致内核调用中断处理程序，进入内核空间。这个过程中，硬件的一些变量和参数也要传递给内核，内核通过这些参数进行中断处理。所谓的“中断上下文”，其实也可以看作就是**硬件传递过来的这些参数和内核需要保存的一些其他环境**（**主要是当前被打断执行的进程环境**）。中断时，内核不代表任何进程运行，它一般只访问系统空间，而不会访问进程空间，内核在中断上下文中执行时一般不会阻塞。