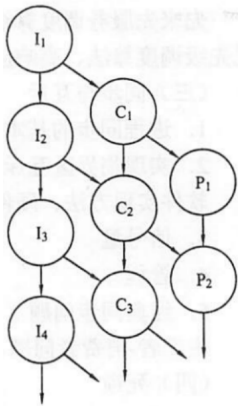
进程管理

1. 进程

1.1. 顺序与并发

进程是OS分配资源，独立运行的基本单位

1.1.1. 前驱图



1. 结点：一个语句/一段程序/一个进程
2. $P_i \rightarrow P_j$意思是P_i必须在P_j开始执行前执行完成
3. 前驱后继，直接前驱直接后继的概念同离散舒徐。没有前驱的叫初始节点，没有后继的叫终止节点

1.1.2. 程序的顺序执行

1. 含义：一个程序分为若干程序，前一个执行完后一个才能执行
2. 特点：顺序性，封闭性(独占资源，结果不受外界影响)，可再现(相同程序执行结果总相同)

1.1.3. 程序的并发执行

1. 含义：无论上个程序是否执行完，下一程序强行执行。导致一定时间内多个程序同时运行且**次序不事先确定**
2. 与顺序执行不同的特点：
    1. 间断性(异步性)：程序走走停停，失去输入时的时序
    2. 失去封闭性：程序得共享资源，运行互相影响，数据可能被乱改
    3. 不可再现性：失去封闭性的结果，例如当两个程序共享一个变量时，运行结果是怎么样不可预测

1.1.4. 程序并发执行的条件

1. 换而言之：让程序在并发执行时保持封闭/可再现性
2. Bernstein条件(理想化条件)

1. 读集$R(p_i)=\{a_1,a_2,\ldots,a_m\}$/写集$W(p_i)=\{b_1,b_2,\ldots,b_m\}$是$p_i$执行所**引用/改变**变量集合
2. Bernstein条件：对于p_1,p_2两个程序，满足以下条件就可再现

   条件1：两次读操作间存储器不变，$R(p_1)\cap W(p_2)=R(p_2)\cap W(p_1)=\varnothing$

   条件2：写操作结果不丢失，$W(p_1)\cap W(p_2)=\varnothing$

PS：串行与并行

1️⃣串行：一个任务执行单元，从物理上就只能执行一个任务，顺序执行在逻辑上也是执行一个任务

2️⃣并行：多个任务执行单元，从物理上多个任务一起执行，并行在逻辑上(一段时间内分时)是多任务但是物理上是单任务(同一时刻不行)

1.2. 进程定义和描述

1.2.1. 进程定义

1️⃣结点定义：是程序在CPU上一次执行过程；是可和别的进程并行执行的计算；程序序在一个数据集合上的运行过程

2️⃣不同OS下的含义：批处理OS中作业=进程(基本认为二者含义相同)。分时OS中进程=用户程序/任务

1.2.2. 进程的特性

1️⃣动态性：创建后产生，调度而执行，得不到资源会暂停，撤销后消亡

2️⃣并发性：可以多个进程都在内存，多个内存同时执行(进程就是因为并发执行才提出)

3️⃣独立性：程是一个能独立运行/分配调度资源的单位

4️⃣异步性：进程以各自独立，不可预知的速度推进

5️⃣结构特征：进程控制块(见后)+程序段(进程中能被调度到CPU上执行的程序代码段)+数据段(初始/中间/执行产生的数据)

1.2.3. 进程&其它概念的辨析

1️⃣进程和程序：进程是程序的执行(动态/静态)，进程是暂时的程序是永久的，一个程序可每次执行出不同进程/一个进程可调用多个程序

2️⃣进程和进程映像：进程映像(实体)是进程某一时刻的静态视图(把进程定住，就是进程映像)，和进程一样都由程序段+数据段+PCB构成

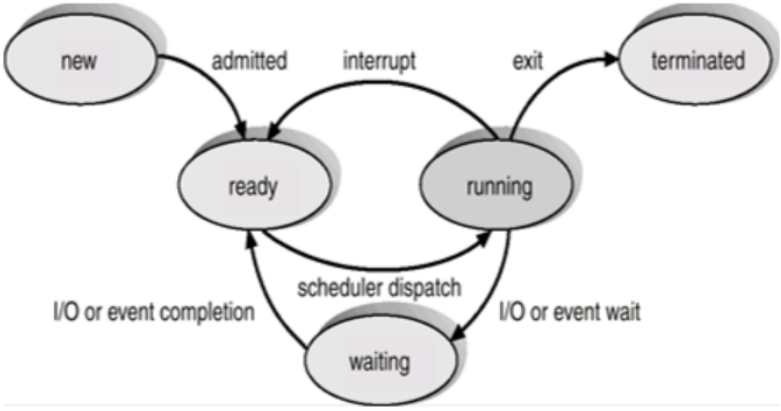
3️⃣进程和作业：

1. 作业是用户向计算机提交任务的任务实体，作业的完成过程为提交+收容+执行+完成。
2. 作业会在外存中排队进入内存，作业进入内存后就是进程(作业是提交后的作业的执行过程)
3. 一个作业有一个及以上进程，一个进程不能有多个作业

1.3. 进程的状态与转换

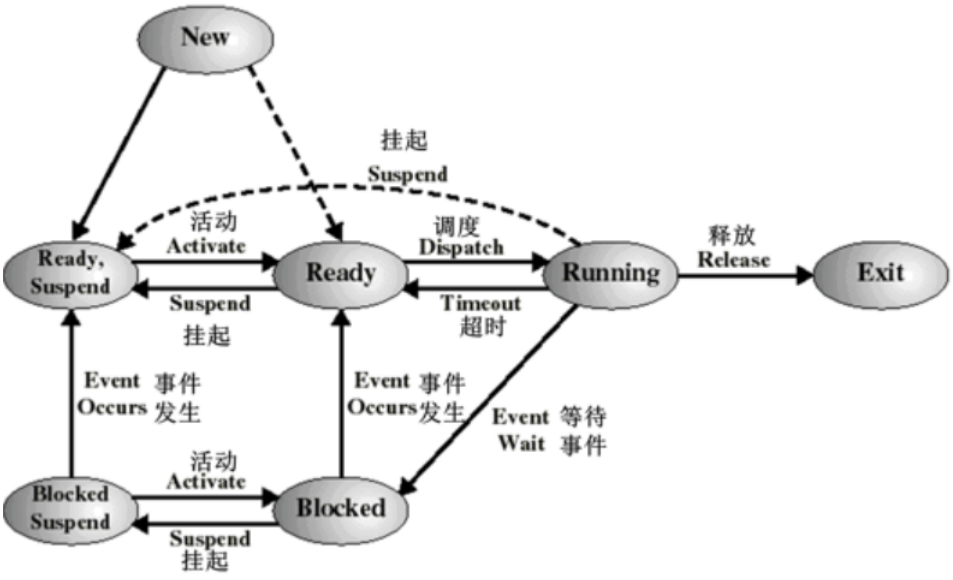
1.3.1. 五状态模型

1. 就绪：已获得除CPU以外的所有资源，CPU时间片没转到这个进程时
2. 执行/运行：获得CPU后在CPU中运行
3. 阻塞/等待：正在执行的进程被打断(如IO完成，缺少数据)，**即使CPU给了进程也执行不了**
4. 创建：申请空白PCB，填写PCB，系统分配资源，最后转入就绪
5. 结束状态：正常执行完/中断退出



1. 就绪→执行：进程被进程调度程序选中
2. 执行→等待(阻塞)：请求，等待某个事件(IO/填补数据)发生完毕
3. 执行→就绪：时间片用完or优先级更高的进程插队
4. 阻塞→就绪：等待到了某此除CPU以外的资源，被唤醒
5. Tips：进程转化不可都逆(上图)，进程间转化并非都是程序主动的(只有执行→阻塞为主动)

1.3.2. 七状态模型：引入了挂起

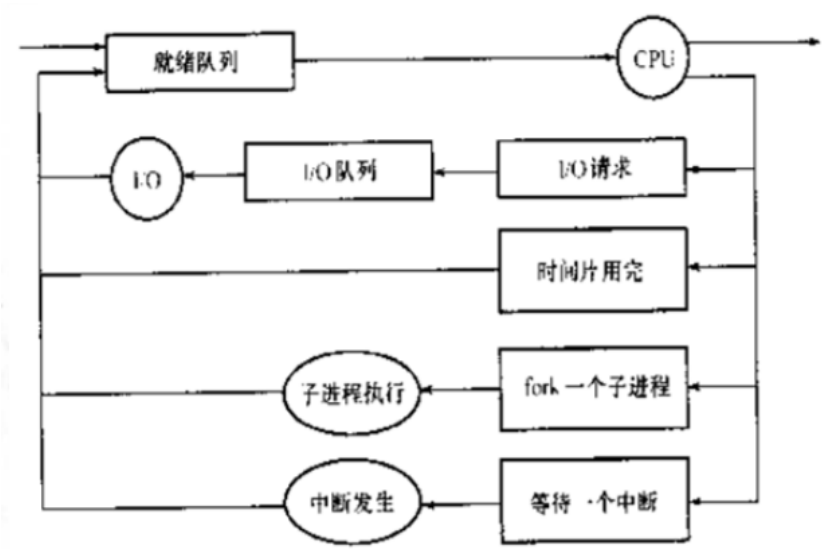


1. 就绪/阻塞挂起：把就绪/阻塞进程从内存丢到外存，保存就绪/阻塞信息，等待重新调入内存

2️⃣引入挂起的意义：优化不活跃的进程，用户可以挂起进程来调查问题，父进程挂起子进程来同步，优化内存等资源利用

1.4. 进程调度

1.4.1. 调度队列



作业队列(所有进程集合)+就绪队列(主存中就绪执行的进程)+设备队列(等待某IO设备的进程)

1.4.2. 三种调度：详见CPU的三级调度

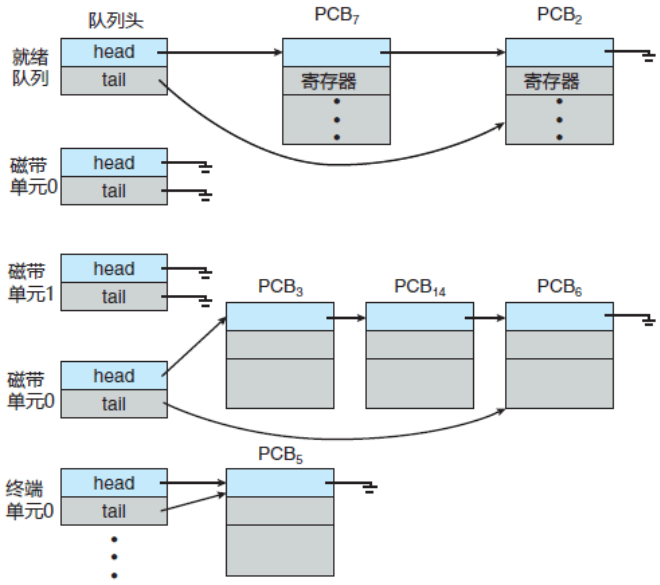
1.4.3. 上下文切换：详见进程切换

1.5. 进程控制数据结构：PCB

1️⃣概述：每个进程都有一个，在进程创建时生成并跟随全程，系统通过PCB识别/感知到/控制/描述进程，常驻内存

2️⃣PCB(进程控制块)的内容

1. 进程标识符(PID)：每个进程唯一持有，进程创建时创建
2. 进程状态：作为进程调度程序分配CPU的依据
3. 进程队列指针：记录PCB队列(如就绪队列/等待队列)中下一个PCB的地址，PCB队列有就绪队列/拥塞队列等



4. 进程的程序与数据的地址
5. 进程优先级(优先级高的可以先被处理器处理)
6. CPU现场保护区：进程脱离CPU时，CPU现场信息(PC，寄存器)被保留到PCB中
7. 通信信息：与其它进程的通信记录
8. 家族联系：比如记录父进程，本进程，子进程，子子进程的关系树
9. 占有资源清单：进程所需/当前已分配资源

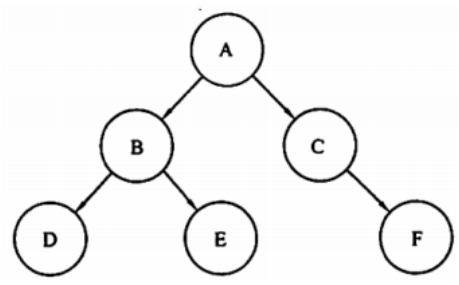
3 PCB的组织方式：链接(PCB链表)+索引(按索引表找PCB)

1.6. 进程控制基础操作

也就是进程管理，由OS内核实现

1.6.1. 创建进程：从程序到进程

1 进程前驱图(进程家族树)：进程创建n个子进程，子进程又创建m个子子进程.....



2 进程创建的诱因：

1. 用户登录(分时OS)：用户在终端输入登录信息，OS为其建立进程然后就绪
2. 作业调度(批处理OS)：作业调度程序让某个作业装入内存，分配资源，变成就绪进程
3. 请求服务：一个进程创建一个子进程，以此类推形成进程树

➕关于父进程子进程

1. 资源共享方式：子进程共享父进程所有/部分/资源+无资源共享
2. 子进程的执行：父子并发执行，父进程等待子进程终止
3. 父子进程的地址空间：两者程序/数据相同，子进程另外加载一个程序

3 进程创建过程：原语，过程为

向OS申请一个PCB(带有PID)→分配资源→初始化PCB(名称/优先级等)→PCB插入就绪队列

4️⃣ UNIX中创建进程的实例

```
int main(int argc, char *argv[])
{
    /*pid存储进程ID，调用fork()系统函数创建子进程*/
    int pid;
    pid = fork();

    /*发生错误，输出错误信息，异常退出*/
    if (pid < 0) {fprintf(stderr, "Fork Failed\n");exit(-1);}

    /*处于新创建的子进程中，子进程的内存空间替换成了"/bin/ls"程序*/
    else if (pid == 0) {execlp("/bin/ls", "ls", NULL);}

    /*处于父进程中，等待父进程的结束，打印子进程结束的信息，正常退出*/
    else {wait(NULL);printf("Child Complete\n");exit(0);}
}
```

1.6.2. 进程撤销

1️⃣ 含义：进程因为完成任务/异常/外界干预，释放各种资源，通过调用exit()返回状态值到OS/父进程

2️⃣ 撤销原语(OS撤销进程的低级操作)

1. 策略：撤销有指定PID的进程，或者顺带其后代进程一起撤销
2. 过程：撤销PCB→停止执行(设置重新调度标志)→回收进程占用资源(给OS或者父进程)

3️⃣ 有关概念

1. 僵尸进程：执行完成且释放资源但仍占据进程表的进程，源于其父进程还未调用wait()读取子进程退出状态
2. 孤儿进程：父进程已经结束(如崩溃)但还在运行(未完成)的子进程，会被init进程，后者定期执行wait()清除这些子进程

1.6.3. 进程阻塞与唤醒

1️⃣ 阻塞/唤醒原语：功能分别为使得进程执行→阻塞，阻塞→就绪(不是执行)

2️⃣ 阻塞/唤醒的原因：

1. 阻塞：当一个进程期待的某一事件未出现时，进程主动调用阻塞原语阻塞自己
2. 唤醒：当一个进程期待的某一事件出现时，发现者进程主动调用唤醒原语，使得阻塞进程被动唤醒

   PS：发现者进程和唤醒进程并发

3️⃣ 阻塞原语的操作流程

中断CPU停止进程→保存CPU现场→进程阻塞并加入等待队列→转到进程调度程序去选一个新进程执行

4️⃣ 唤醒原语的操作流程：将被唤醒进程从等待队列中移出→就绪→插入就绪队列

1.7. 进程控制其他操作

1.7.1. 进程切换/上下文切换(开销较大)

1. 含义：处理器从一个进程的运行转到另一个进程的运行
2. 进程切换：CPU将旧进程上下文保存在PCB中→加载新进程上下文
3. 这一过程中会产生中断，CPU会从用户模式→内核模式→用户模式

1.7.2. 协同进程

1. 独立/协同进程：是(协同)否(独立)需要与其他进程共享信息或在执行上相互作用

PS: 进程是否可以进行进程通信和他是独立/协同没有必然联系

2. 协同进程的好处：信息共享，模块化，加速运算

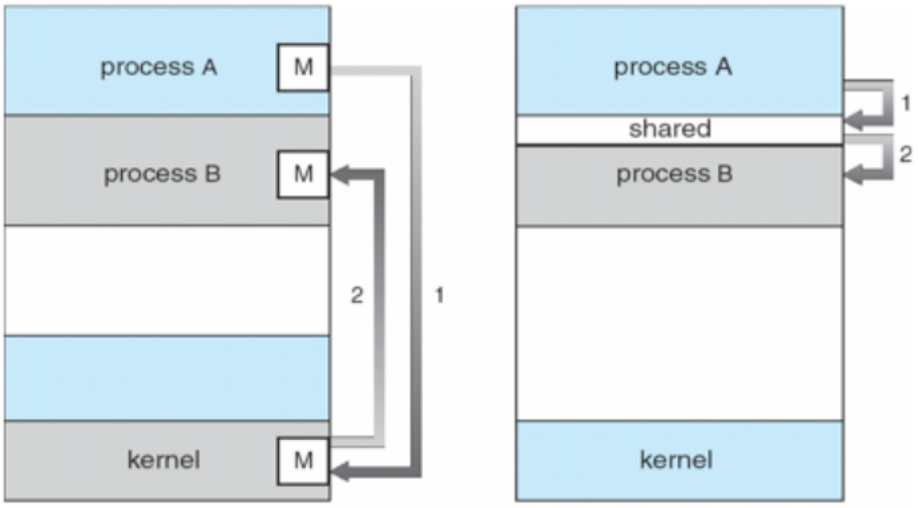
1.7.3. 进程通信⬇

1.8. 进程通信(IPC)

1.8.1. 低级进程通信

其实就是进程的同步P和互斥V，对应的P/V原语为低级进程通信原语

1.8.2. 三种高级进程通信



1. 共享存储系统
    1. 含义：主存中的共享区域，多个进程通过对这个区域读写来实现通信
    2. 特点：由通信进程确定交换的数据和位置，**不受OS控制**
2. 消息传递系统
    1. 含义：进程通过建立通信连接(物理的总线/逻辑的程序)，通过send/receive交换信息
    2. 直接通信方式：发送进程发信给接收进程→消息进入接收进程的缓冲队列→接收进程从队列中取得消息
    3. 间接通信方式：创建邮箱→发送进程把消息给信箱(端口Port)→接收进程从邮箱取得消息→销毁邮箱(Optional)，邮箱分为私有(进程创建的)/公有的(OS创建的)

3️⃣管道通信系统

1. 管道：连接读写进程，实现二者间通信的共享文件，并不是一个传输通道
2. 过程：向管道提供写进程，信息以字符流形式送入管道，而接收管道通过读进程输出

1.8.3. 消息传递的异步/同步

1️⃣零容忍/同步/阻塞：发送者必须等待接收者

2️⃣有限容忍：接收缓冲队列里达到n长后，发送者就必须等了

3️⃣无限容忍/异步/非阻塞：发送者一直发送消息不等待，接收者同样不等待

2. 线程

2.1. 线程的引入

1️⃣早期OS进程的基本属性：拥有资源的独立单位+可调度的基本单位

2️⃣弊端：进程拥有资源又要频繁调度，开销大，限制了并发

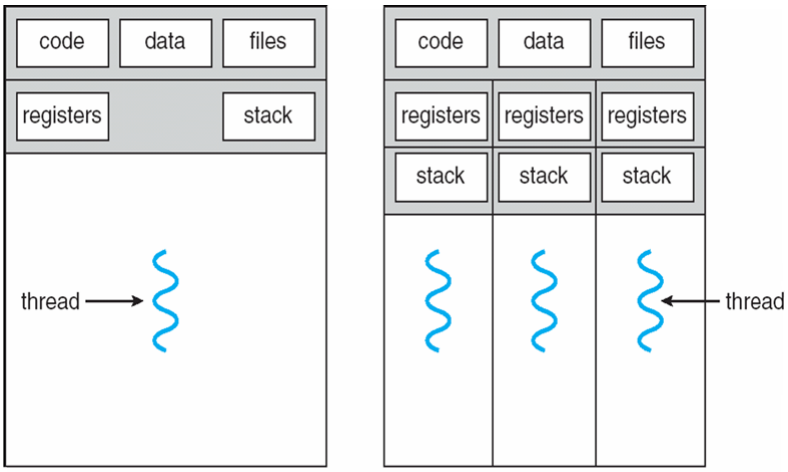
3️⃣解决方案：让进程成为拥有资源的单位，不频繁切换；让线程成为调度单位，不拥有资源

2.2. 线程的概念

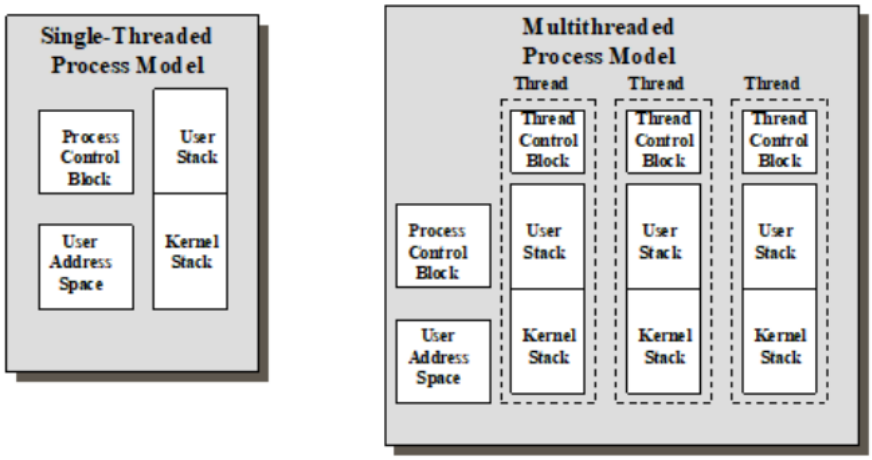
2️⃣线程定义：作为CPU调度单位(进程只作为资源分配单位)，aka轻型进程

3️⃣线程的资源：只拥有必不可少资源(线程状态+程序计数器+寄存器上下文+栈)，但是和同属一个进程的线程们共享资源(代码段+数据段+OS资源)

4️⃣多线程：一个进程有多个线程并发执行，一线程改了数据其他线程也使用修改数据， 一线程读文件时其他线程也可同时读



single-threaded process　　multithreaded process



5️⃣线程切换：速度极快，只需切换寄存器+栈

2.3. 进程VS线程

1️⃣调度：耗时

同一进程内切换上下文<进程切换上下文=不同进程的线程切换上下文(需要进程切换)

2️⃣拥有资源：进程拥有资源，线程只拥有必要资源(线程状态+程序计数器+寄存器上下文+栈)不拥有系统资源

3️⃣并发性：引入线程的OS中，进程可并发，同一进程的线程也可并发

4️⃣系统开销：线程切换是涉及少量寄存器内容，开销很小；进程切换需要分配/回收资源

5️⃣通信：线程通信不用OS干预，通过读写进程数据段通信

2.4. 线程的优点

1️⃣响应度高：多线程中即使某线程阻塞，其他线程还可以顶上，不用等待

2️⃣经济性：并发执行的时空开销小(线程建立/终止/切换耗时短)，由于共享进程资源故可以减少通信频率，有助于提高并发度

3️⃣多线程更利于多处理器(MP)架构

2.5. 内核线程与用户线程

2.5.1. 核级线程

1️⃣含义：由OS内核创建/撤销的线程，存在于在支持内核级线程的OS中，此时CPU调度的是线程

2️⃣特点：

1. 内核维护进程和线程的上下文信息并完成线程切换
2. 内核级线程IO操作阻塞时，不影响其他线程
3. 处理器时间片分配对象是线程，多个线程的进程将获得更多处理器时间

2.5.2. 用户级线程

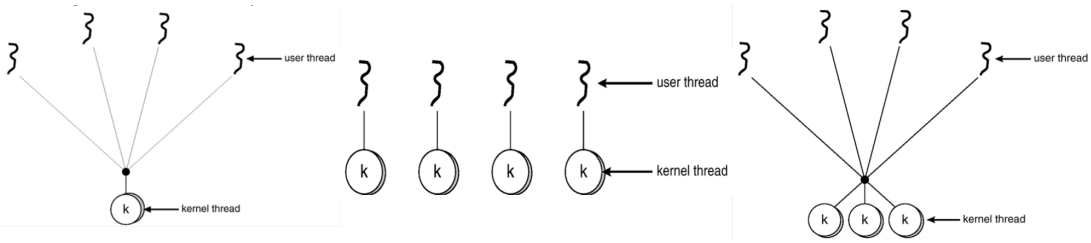
1️⃣含义：由用户级线程库管理的线程，线程库创建/管理线程(无需内核)，此时CPU调度的是进程

2️⃣特点：

1. 用户级线程切换不需要内核特权
2. 其调度在应用进程内部进行，可针对应用优化调度，调度过程简单快速(无须用户态/核心态切换)
3. 缺点在于OS不知道线程的存在，一个线程阻塞时整个进程都等待
4. 处理器时间片分配给进程，进程多线程时，每个线程执行时间减少

2.6. 多线程模型

首先明确一点：用户级/内核级线程通常在同一进程进行映射/管理



1️⃣多对一(不可并发/开销小)：多用户级线程映射到一内核级线程

1. 优点：线程在用户空间管理，效率高
2. 缺点：OS只能识别那个内核级线程，一个内核线程只能执行其中一个用户进程，一个用户级线程堵住进程就会堵

2️⃣一对一(可并发/开销大)：一用户级线程映射到一内核级线程

1. 优点：一线程阻塞不影响其他线程，可以多线程并行
2. 缺点：创建用户线程必须创建内核级线程，开销大

3️⃣多对多(可并发/开销相对小)：多用户级线程映射到多内核级线程(内核级线程数不多于用户级线程数)，多对多模型允许真正并行，打破用户级线程限制，优化阻塞与调度

2.7. 线程锁

锁的功能越强大，性能就越拉跨

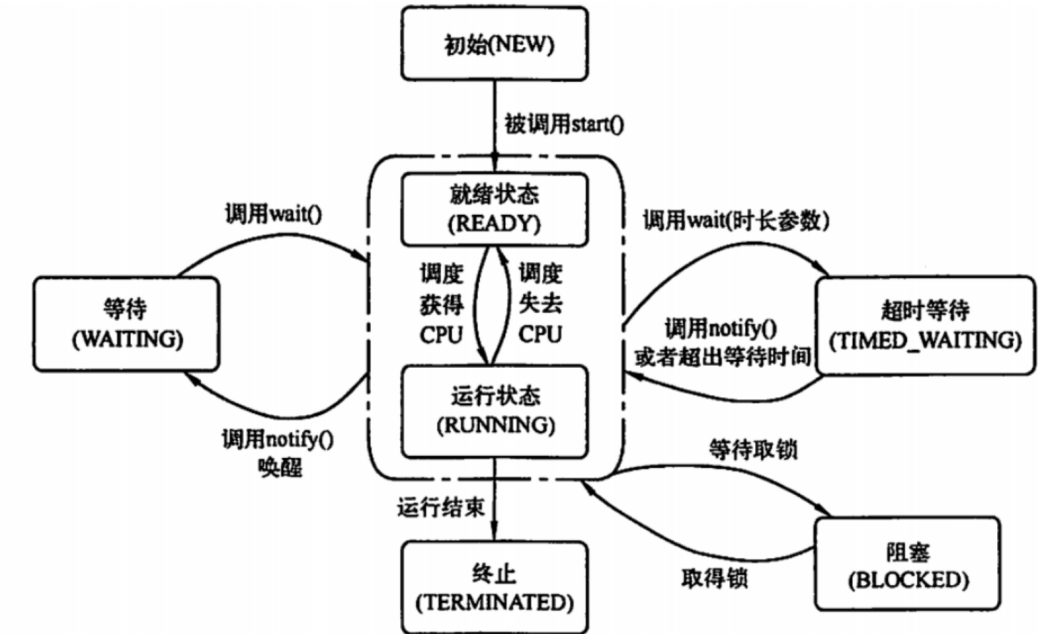
1️⃣互斥锁：信号量，确保同时只有一个线程能够访问特定的资源，一个进程有锁其它的都等待

2️⃣条件锁：条件变量，允许线程在特定条件(互斥锁大哥同不同意)暂停/继续

3️⃣自旋锁：与互斥锁类似，区别在于无法获得资源时，互斥锁会让线程滚/自旋锁会让线程等(不断检测锁是否可用)

4️⃣读写锁：允许多线程同时读共享资源，但只允许一个线程写

2.8. 线程的生命周期(状态转化)



1️⃣初始：创建线程

2️⃣就绪：创建后，通过其它运行线程调用start()方法，加入可运行线程池(就绪)

3️⃣运行：就绪+CPU开始运行，2️⃣+3️⃣也称可运行状态

4️⃣阻塞：线程放弃CPU使用权，暂停，可自动唤醒

5️⃣等待：线程调用wait()方法，进入等待队列释放占用资源，不可自动唤醒(依赖其他线程调用notify()方法)

6️⃣超时等待：与等待的区别仅在于超时等待是等一段时间，等待是一直等

7️⃣终止：线程执行完了

PS：概念补充

PS.1. 线程池

1️⃣含义：一组预先初始化的线程

2️⃣工作方式：要执行某任务时免去建立线程的开销，直接调用线程池中的空闲线程去执行，执行完后也不销毁又丢回线程池

3️⃣好处：资源消耗小，响应快

PS.2. 线程库

1. 含义：是程序员创建和管理线程的API
2. 分类：用户级线程库(POSIX Pthreads)+内核级(Win32 threads)+其他(Java thread)