**操作系统**

1. **进程管理（cpu）**
2. **存储管理（内存）**
3. **设备管理（外设）**
4. **文件（外存）**

单道批处理系统（道：运行在系统中进程或者的数量，批：外存外有一批作业在等待，进到内存只有一个，就是单道批处理系统）

多道（多个进程进入到系统中抢夺资源，如何抢夺资源需要我们去研究）

分时系统（分时的运行多个线程，一个线程运行消耗一毫秒，然后切换到下一个线程）

并发vs并行

操作系统一般的体系结构

执行的模式（单模式、双模式、多模式）

微核结构

**什么是操作系统：**

关于操作系统的四个基本观点

外部看os：计算机用户的观点：用户环境观点（用）

应用程序员的观点：虚拟机器观点（把机器虚拟化）

内部看os：

os开发者观点之一：资源管理观点（课程观点）

os开发者观点之二：作业组织系统

1. 用户接口系统（为用户提供方便的环境）
2. 虚拟机：为应用软件提供了许多比计算机硬件功能更强或计算硬件所没有的 功能，通过虚拟机扩展硬件的上限。
3. 资源管理器：

负责分配、回收以及控制系统中的各种软硬件资源（进程、cpu）

1. 作业组织：

协调系统中运行的各个应用软件的运行次序。（作业：请求）

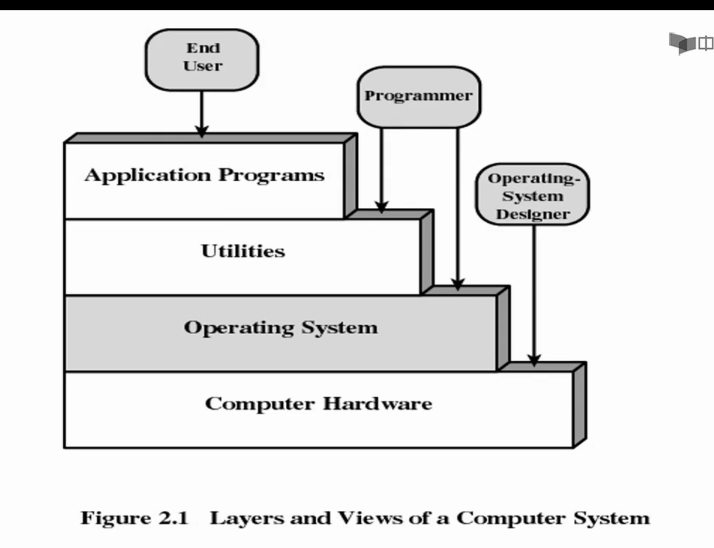
现代操作系统的功能和非功能性需求

操作系统的发展、类型和特征

现代操作系统体系结构基础知识

硬件—》操作系统—》开发工具—》应用程序

操作系统设计者—》程序员（基于操作系统、基于工具IDEA类型）—》终端用户



**1.操作系统的需求：**

**需求分析-》系统设计-》编码设计**

**系统需求是：人们从软件系统的外部对软件系统提出的诸多期望。这些期望包括三种类型：**

1. 软件系统能提供的服务；
2. 软件系统在提供这些服务时，需要满足的限制条件；
3. 系统具有适应某些变化的能力；（扩展、移植）

第一类系统是前两类系统需求赖以生存的基础，称之为软件系统的功能性需求，后两类系统需求称为软件系统的非功能性需求。

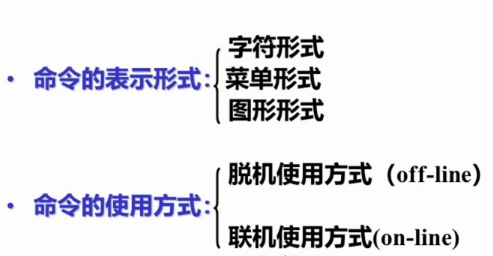
**1.2.1os的功能性需求：**

计算机用户需求的用户命令：由os实现的所有的所有用户命令构成的集合被人们称为os的interface（用户接口）；有时也被称为命令接口

应用软件需要的系统调用（System call）： 由os实现的所有系统调用所构成的集合被人称为程序接口（API）



用户命令：指计算机用户要求计算机系统为其工作的指示



脱机和联机，是否在操作系统的控制下运行的，是则是联机的，不是就是脱机的。

系统调用：指由os实现的应用软件在运行过程中可以调用的系统服务。

（两种常用的API:POSIX.1、WIN32 API）

程序接口事实上定义了一台虚拟计算机。该虚拟计算机包含了一组抽象概念以及与这组抽象概念相关的系统服务。

**1.2.2os的非功能性需求：**

性能和效率

吞吐量（单位时间系统完成的任务，最少时间完成尽量多的任务）

响应时间（点了马上出来，程序需要干嘛，cpu做中断分析，是否中断，怎么中断， 中断后的收尾工作）

分时系统满足尽量多的任务（在有限的硬件资源，通过os配合硬件系统最大化用户数）

公平性：设计出来的算法不能偏向某一应用或某一线程（不同用户要求不同，例如无人汽车强调及时响应）

可靠性：系统稳定

安全性：

可伸缩：根据要求购买不同配置

可扩展性：新的外部设备的增长。（手机来了，电脑和手机相连）

可移植性：不同硬件、平台下都可

**不同系统对需求不同**

**1.2.3os对硬件平台的依赖**

IO中断支持（支持人机交互的基础）是硬件的支持，Timer（计时器）、DMA（直接存储器访问）（存储，当**dma**达到一定程度才提交cpu中断，有关），特权指令（由硬件支撑）、存储保护的机制（检查有没有越界，**地址的转换：把虚拟值转换成计算中的实际物理地址、**MMU（存储管理单元）硬件的支撑）

**1.2.4基本概念：**

作业：计算机用户在上机过程中要求计算机系统为其所作工作的集合；作业中每项相对独立的工作称为作业步。通常人们用一组命令来描述作业，其中、每个命令定义一个作业步。

（分为联机：作业过程，用户可改变作业步 脱机：用户无法控制，一旦开始）

进程和线程的概念：

线程是**程序一次相对独立**的**运行过程**（动态），在现代os中，线程是系统调度的最小单位，有资源。 （资源调度问题）

进程：是**系统分配资源的基本对象；**在os中，进程仅仅是系统**拥有资源的最小实体**；不过在传统os中，进程同时**也是系统调度的最小单位**。（分配问题，资源调度问题）

线程归属某些进程，由进程获取资源，线程可以使用，但调度获取资源时，由进程来进行。

虚拟存储：

进程的逻辑地址空间（内存或者外存）；多级物理存储体系的高度抽象结果。

将实际的内存空间虚拟扩大，由外存（把外存虚拟为内存使用）来支撑。

文件：

命名了的字节流（你就能说这段二进制数字就是文件）

**1.3操作系统的演变的原因（系统不断升级）**

**修复错误、打补丁（因为是人设计的规则，不断增加规则防止黑客）**

**新的服务（新设备、提供的服务提供资源的支持）**

**硬件升级（不同硬件升级，需求适配，驱动本身不属于操作系统（生产商根据系统商的标准接口实现驱动程序））**

**性能要求（硬件做不到，软件来做，比如单核模仿多线程，通过并发）**

**串行处理：**

**原始：无操作系统，机器通过拨按钮来运行，将浪费机器的运行空间；**

**启动很慢，把代码啊，编译器手动拼接。**

**单处理批系统（第一代系统）：（为了解决如上问题）**

**有一批作业在外存等待，要到内存中运行，一次只能进来一个作业，谁把作业外存提交到内存，Monitor（监视器常驻内存，当内存没有作业时，就会在外存选一个，那选哪个（作业调度问题），而作业完成主动通知monitor，退出。如果作业有IO会占用很长时间，所以考虑多道）**

**单道程序设计：**

**而Monitor就是单道程序设计：处理器通常比IO快，所以CPU就要wait，所以还是低效的，这段时间的cpu资源是浪费的。**

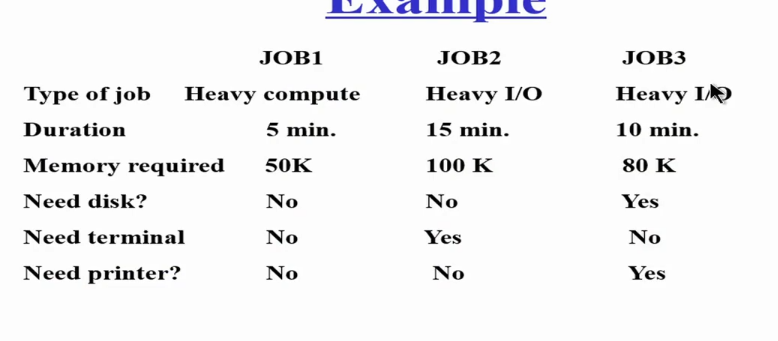
**多道程序设计：**

一个支持mutiprogramming（多道程序设计）的系统允许多道程序**同时准备运行；当正在运行的那道程序因为某种原因（比如等待IO）暂时不能继续运行时，系统将自动地将启动另一道程序运行；一旦原因消除（比如数据到达或数据已经输出完毕），暂时停止的运行的那道程序在将来某个时点可以被系统重新启动继续运行。**

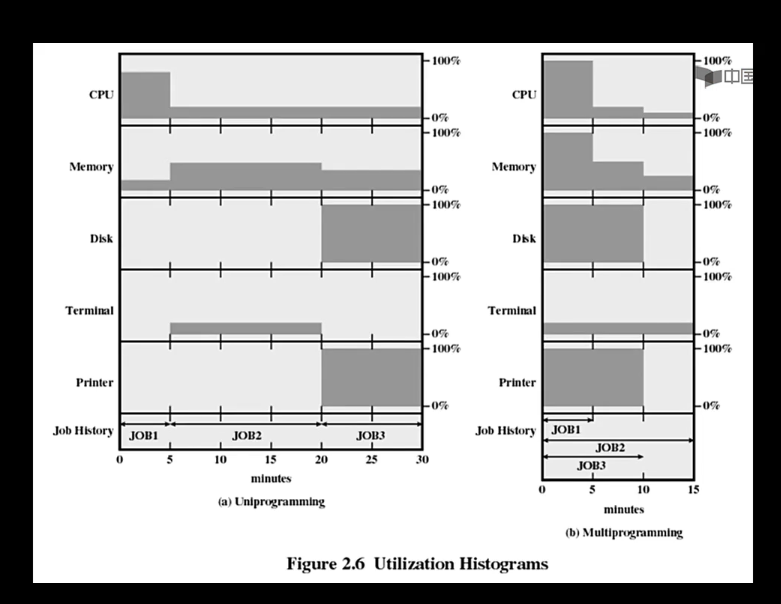
**允许多个任务到内存空间（涉及到内存划分，cpu资源问题，抢内存资源，抢cpu资源），但到内存还是要排队（涉及到调度）进入准备状态。同时失去了封闭性**

（微观上是交替运行，等待原因；IO、通信）太多任务将导致效率大大降低，故需要研究多少进程才合适。

**多道批处理系统实例：**

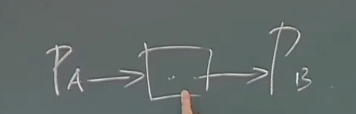


合理系统是IO和计算混合的。



多道系统的问题：

资源竞争问题，同步指的是一种协调（计算机A，和打印机B，B要求A数据时，A尚未计算出，需要等待；当计算出数据，通知B，打印数据。计算出的数据放在存储区中）若计算速度太快，容易造成空间溢出，则需要判断不为满才放数据；而B需要判断空间是否有数据，有才取。



**异步**：进程以不可预知的速度向前推进。（何时执行、何时暂停、时间、速度 不可预知，进程间难以互相协调，同时执行的结果多种多样）

**同步**：为完成某种任务而建立的两个或多个线程，凭借同步机制来协调他们的 工作次序而等待、传递消息。

**互斥：**比如说共享区正在写，你也写或者读，不允许，现在拿到的是脏数据。都写数据，分先后，互斥。

**操作的不确定性** ：内存中存在多个作业，相互之间会干扰，会影响到结果， 容易造成不确定性。

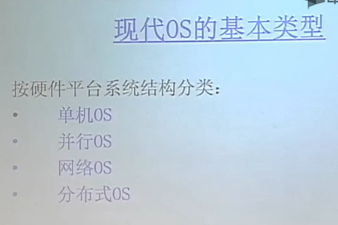
**死锁**：大家都抢着资源不释放。

**分时系统**（Time Sharing多道系统，对每个任务（还是得排队）提供固定的时间切片，非常适合交互式系统分时处理大量用户请求作业，然而任务过多，每运行完一次就中断一次，中断耗费时间，降低效率，而且有时候不能中断，中断后无法回复）

**多道批处理系统和分时系统的比较：**

多道系统：最大化利用处理器，作业通过作业控制管理语言提交命令。

分时系统：响应时间尽量短，用户通过终端提交命令。



单机OS的基本类型：

功能特征分：

批处理系统（吞吐量大）

分时系统（对多用户交互，响应快，响应时间短）

实时系统（实时信息系统（抢票），实时控制系统（无人汽车），要对数据库的数据库及时更新，有机制更新）

两个基本特征：

任务共行：

宏观上看，任务共行是指系统有多个任务同时运行

微观上看，任务共行指**单处理机系统中的任务并发**（多个任务在单个处理机上交替运行）或**多处理系统中的任务并行**（多个任务多个处理机上同时运行）

资源共享：

宏观上：资源共享是指多个任务可以同时使用系统中的软硬件资源

微观上：资源共享是指多个任务可以交替互斥地使用系统中的某个资源

任务管理模型：

task是指：计算机系统在某个资源集合上所做的一次相对独立的**计算过程**。

现代os的任务中，用线程和进程两个概念共同表示

传统os中，任务仅仅用进程这一基本概念表示

现代os中，任务管理模型用**线程状态转换图表示，在传统os中任务管理模型用进程状态转换图表示。（确定该任务在某时点在干嘛，状态是什么，是一个状态变换的过程）**

资源管理模型：

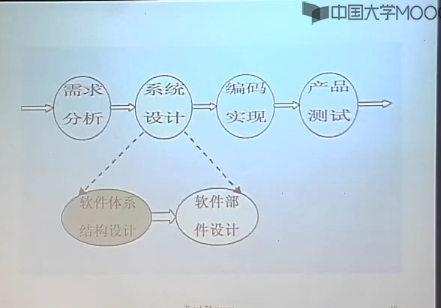
资源：由程序和数据组成的**软件资源**以及包含CPU、存储器、I/O设备等在内的**硬件资源**。

通常情况下，系统用**竞争模式（打开某个软件、数据表、以竞争的方式）**管理软件资源：为此系统将为共享同一软件资源的多个任务提供**互斥**机制。

对于硬件资源，系统常常采用**分配模式**加以管理，该模式可以描述为：

申请-》分配-》使用-》释放-》回收

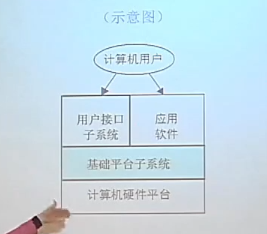
1.4 OS架构



1. 一种常见的os**总体结构**风格

大多数现代os其总体结构包括两类子系统：一是用户接口子系统，二是基础平台子系统。其中，用户接口子系统提供计算机用户需求的**用户命令（用户使用）**，其基础平台子系统提供软件需求的**系统调用（可用于开发其他软件）**。

用户接口子系统与基础平台子系统之间的相互关系具有**单向性**。（接口子系统实现用户命令时**能够引用基础平台子系统提供的各种系统调用**，但基础平台子系统在实现各种系统调用时**不会引用用户接口子系统提供的各种用户命令，简单说就是用户接口子系统是由基础平台子系统支撑的**）（用户接口上层，基础平台子系统底层（提供用户使用硬件的桥梁））



1. OS基础平台子系统结构风格（更关心）

**分层结构风格：（高严格）**

类型osi七层的概念，使用分层结构的基础平台子系统结构包含若干层，每一层实现一组基本概念以及与其相关的基本属性。哪一层实现哪些功能。

层层之间的相关相互关系：

所有各层的实现不依赖其以上各层提供的概念及其属性，**只依赖其直接下层**所提供的概念及属性（不能越层，层的划分十分严格，每层均对其上各层隐藏其下层的存在）；

**分级结构风格：（中严格）**

使用分级结构的基础平台子系统结构包含若干级，每一级实现一组基本概念以及与其相关的基本属性。

所有各级的实现不依赖其以上各级所提供的概念及属性，只依赖以下**各级**所提供的概念以及属性。（各依赖所有下级，下级均可见，条件相比分层结构更为宽松）（过于严格的结构将导致实现困难提高，有时需要降低要求）

**分块结构风格：（低严格）**

使用分块结构的基础平台子系统结构包含若干**模块**，每一块实现一组基本概念以及与其相关的基本属性。

**所有各块均可以任意引用其他各块所提供的概念及属性**

**分层、分级、分块结构的关系：**

**分层结构风格是一种特殊的分级结构风格，分级结构是一种特殊的分块结构。**

**（分块结构 > 分级机构 > 分层结构）（linux就是分块结构）**

**分层有利于**实现基础平台的可维护性、可拓展性、可移植性、部件可重用等非功能性需求。**不利于**提高基础平台子系统的时间和空间效率（严格的分层需要逐层调用下层的功能，故时间空间效率低）；构造一个纯粹的分层结构将非常困难。

分块结构是一个切合实际的做法；**有利于**生成高效、紧凑的基础平台子系统可执行代码；**不利于**实现基础平台子系统的灵活性

相对于分层和分块，**分级结构风格的长处和不足介于两者**之间。

1. 双模式基础平台子系统结构风格

**模式结构风格：**

**模式，就是程序在运行过程中使用的、由硬件结构提供的CPU特权模式。**

**多模式结构风格：**

使用多模式结构风格的基础平台子系统结构包含**多个**模式模块；这些模式模块或者是一个应用软件或者是基础平台子系统的一部分。（意味着性能下代价）

**单模式结构风格：**

使用单模式结构风格的基础平台子系统结构仅仅包含**一个模式模块；**该模式模块由应用软件和基础平台子系统共同组成。

在使用单模式结构风格的基础平台子系统结构中，应用软件和基础平台子系统在同一cpu特权模式下运行。（例如DOS系统）

**比较：**

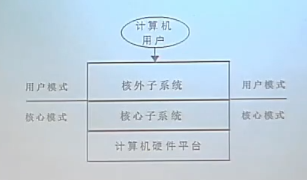
**多模式结构风格**有利于基础平台子系统的**可靠性、安全性等非功能性需求**；会降低基础平台子系统的**性能**；（用户模式修改系统内部数据，系统需要判断并且切换模式即切换进程，这一过程会造成性能的开销）在较高级别CPU特权模式下**调试程序是一件困难**的事情。（用户切换到高级模式往往需要花很长时间，很难追踪故障和bug）

**单模式结构风格**不会增加基础平台子系统的**开发难度；**不会影响基础平台**子系统的性能**；不利于实现平台子系统的**可靠性、安全性等非功能性需求**。

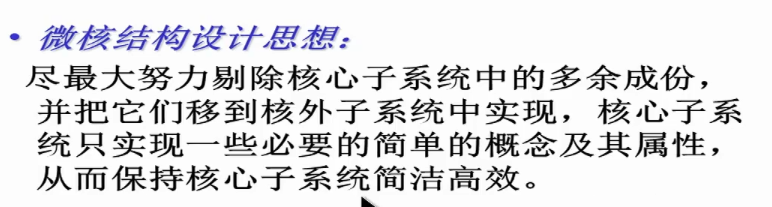
而这两者的折中方案就是**双模式基础平台子系统：**

双模式基础平台子系统其总体结构包含两个模式模块；它们分别在两种不同的cpu特权模式下运行（**核外子系统和核心子系统**）；把核外子系统使用的CPU特权称为**UserMode**,把核心子系统所使用的CPU特权模式称为**KernelMode （用户模式（**用户模式的程序的数据可相互调用**）、**

**核心模式（**遇到系统调用时，要切换到核心模式，还要对进入核心模式的操作进行判定（进入系统干嘛，各种规则，黑客就是发现规则漏洞，修漏洞就是补规则），判定非法则中断就是**软中断**））



1. 微核结构



实现非常困难，界定核心问题，但分布式系统用的特别多。

# 进程（描述程序的执行过程，动态的）

程序的顺序执行和并发执行：

顺序执行：顺序性、封闭性（一个程序在系统里占用了系统的资源的，只与它的输入有关，在封闭的环境里面执行），可再现性（多次执行结果都相同）

**并发执行**：间断性，非封闭性（进程之间存在，互相争夺资源）、执行结果不可再现（存款的同时存取，结果可能不一样）

**程序并发执行条件（Bernstein条件）**

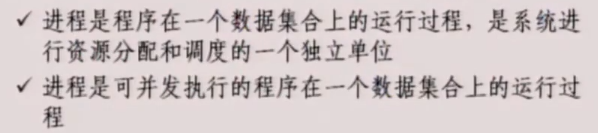
P1、P2表示任意两个进程，R表示读，W表示写



意思就是：两个线程的读和写，同时写不能同时进行（读完再写，写完再读（事务处理）），但可以同时读。

**进程的概念与特征：**

进程可以看作一个任务



进程可以被跟踪的（根据执行指令的序列来跟踪）

特点：动态性（执行、停止动态的一个过程）、并发性（多进程交替执行）、独立性（相互独立），异步性。

**进程的结构：**

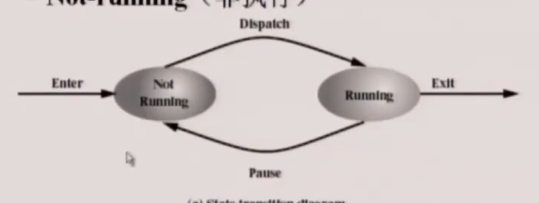
**程序、数据、PCB（进程控制块）**（进程依赖程序，程序运行必须有一定的数据，而进程拥有自己的信息和操作就是PCB ）

**进程的状态：**

**进程两状态:**

**Running(执行)**

**Not-Running（非执行）**



注：并非所有进程只要Not-Running就处于ready状态，有的需要Blocked等待I/O完成。

**5状态：**

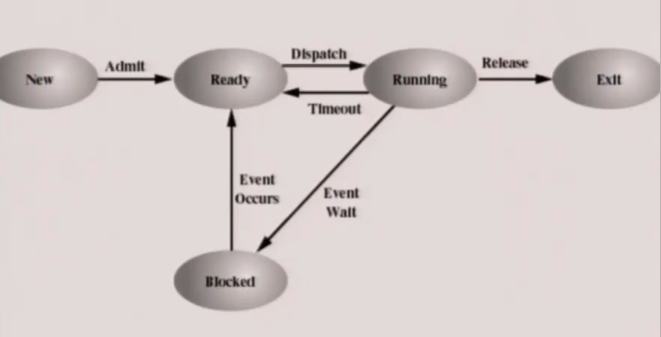
Running：占用处理机，通过调度才能进入Running状态

Ready：准备执行状态

Blocked：等待某事件（IO、等待时间）发生才执行（需调度）的状态（会发生）

New：进程已经创建，但未被OS接纳为可执行进程

Exit：因为停止或取消，被OS从执行状态释放（此时进程属于退出状态还属于一个进程吗？不占用资源，但又还存在，僵尸进程）



**5状态的转换：**

Null——》New ：新创建线程首先处于新状态

New——》Ready：OS接纳新状态进程为就绪进程

Ready——》Running：OS只能从就绪进程中选一个进程执行（调度器dispatch）

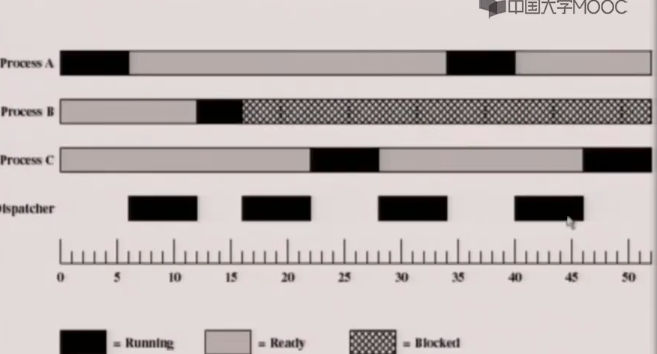
Running——》Exit：执行状态的进程执行完毕，或被取消，则转换为退出 状态。

Running——》Ready：分片系统中，时间片用完，或优先级高的进程到来， 将终止优先级低的线程的执行

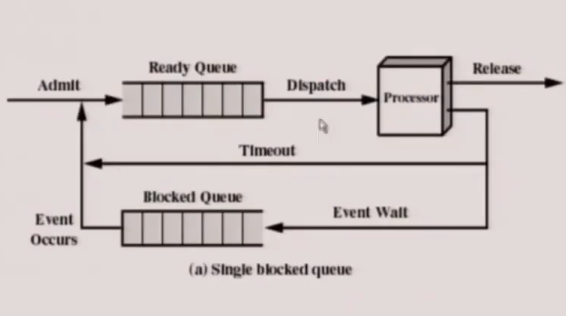
Blocked——》Ready：当阻塞进程等待的事件发生，就转换为就绪状态。

Ready——》Exit：某些系统允许父进程在任何情况下终止其子进程。若一个父进程终止，其子孙进程都必须终止。

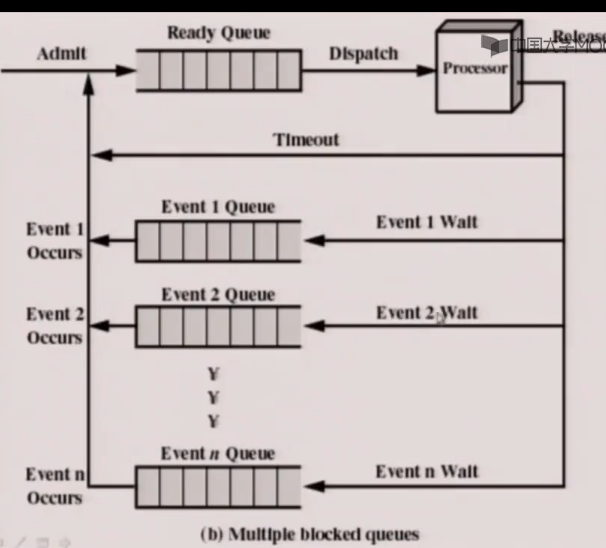
Blocked——》Exit：同前



交替执行，显然进程切换将耗费资源，降低效率

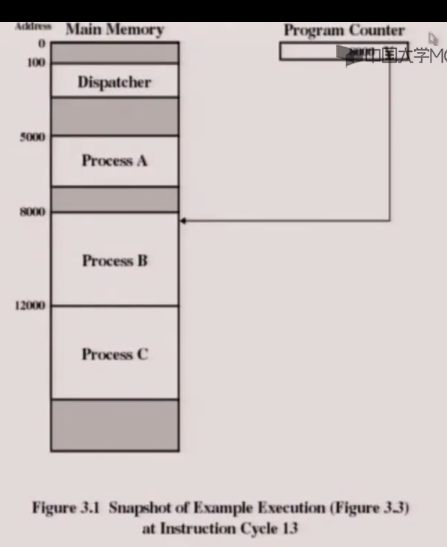


单堵塞队列进程

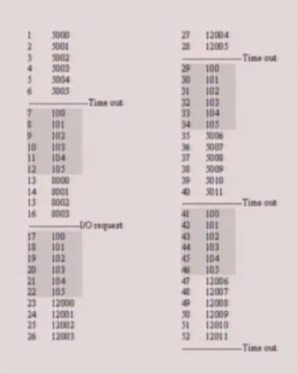


多堵塞队列进程（阻塞原因不同，进入不同的阻塞队列，堵塞状态的进程不能被调度）

故Not-Running分为ready和Blocked



进程中dispatcher就类似monitor（监视器），可看作A先进，BC后进排成队列。Program Counter程序计数器拿来记录当前程序执行到了第几条指令



（数字表示线程指令在内存中的地址）

1.TimeOut：5000-5005，进程A执行了6条指令时间到就中断了（时间超时）

2.100-105，调度程序执行，调度下一个线程B执行

3.8000-8003线程B执行，I/Orequest，I/O中断

4.100-105 调度程序执行，调度线程C执行

5.12000-12005线程C执行，Timeout，时间中断。

6.100-105，调度程序执行，调度线程A执行

7.5006-5011，线程A执行，时间Timeout，中断

8.100-105 调度程序执行，调度线程C执行（因为B还在I/O状态中）

9.12006-12011 线程C执行

（调度程序属于系统进程，每次使用调度程序，有进程切换，模式切换到系统中执行）

**交换技术（Swapping）**

将内存中暂时**不能运行的进程**，或**暂时不用的数据和程序**，swapping-ouut到**外存**，以腾出足够的内存空间、把**已具备运行条件的进程**或**进程所需要的数据和程序**，swapping-in内存。

**suspended processes（挂起进程）（基础支撑，交换技术，交换过程实际就是一个IO过程）**

CPU比IO快，所以CPU总是等待，为解决这个问题：

交换这些进程到外存来释放更多的内存空间，当进程被交换到外存上会变成挂起状态。

第6状态：

**挂起的原因：**

1. 执行IO等其他等待事件挂起进程释放内存空间
2. 其他系统原因：周期性的系统软件（比如说守护进程，检查死锁等等，周期没 来挂起）
3. 交互式的用户请求可以挂起进程
4. 基于时间中断
5. 父进程请求挂起进程

被挂起进程的特征：

1.不能立即执行

2.可能是等待某事件发生。若是，则阻塞条件独立于挂起事件（挂起是由于其他原因造成的），即使阻塞事件发生，该进程也不能执行（阻塞解除，对挂起的原因没有影响，挂起和解除挂起是一对操作，而不是阻塞解除就解除挂起）。

3.使之挂起的进程为：自身、其父进程，OS。

4.**只有挂起它的进程才能使之由挂起状态转换为其他状态。**

问题：是否能挂起阻塞？

挂起是基于交换技术，一定是把它的程序和数据放到外存去了。那么程序数据回来就是解除挂起。

不一定能挂起阻塞进程，但很多时候优先挂起阻塞进程，因为外存等待和内存等待一样。

如何激活一个挂起进程？

就是把它拿出去的数据一个IO的过程。

区分两个概念：

进程是否等待事件（等待事件就是堵塞），阻塞与否

进程是否被换出内存（换出内存就是挂起），挂起与否

**4种状态的组合：**

Ready：进程在内存，准备执行

Blocked：进程在内存，等待事件

**（新状态）Ready，Suspend：**进程在外存，只要调入内存即可执行。（把它从外存拿进来，进入就绪队列）

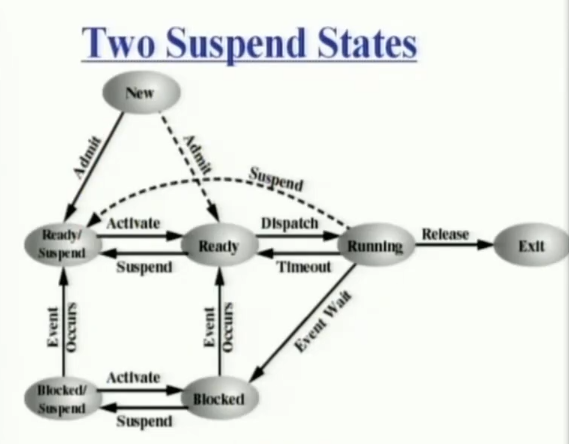
**（新状态）Blocked，Suspend**：进程在外存，等待事件（把它从外存拿进来，进入阻塞队列）

处理机可调度执行的进程有两种：

1. 新创建的进程
2. 或换入一个以前挂起的进程

通常为避免增加系统负载，系统会换入**一个以前挂起的进程执行，可缩短周转时间**。

**具有挂起状态的进程状态转换：（进程七状态转换）**



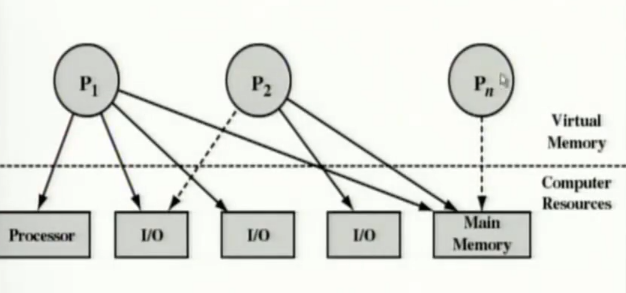
**New-》Ready/Suspend：**原因是内存不够，需要放在外存，不能被执行，一旦有内存就马上进去。

**Blocked—》Ready/Suspend:：**IO时，cpu从外存请求磁盘数据到缓冲区，此过程在进行，此时进程被挂起，当数据已经读入到缓冲区了，就会从Bocked变为Ready态，而此时的挂起状态并没有被通知解除挂起，就变成了Ready，Suspend。（那么此时这个状态的改变是谁改？OS控制，那些状态信息在哪呢？PCB，PCB存放线程的信息）

Running—》Ready，Suspend：当执行进程的时间片用完时，会转换为Ready，或一个高优先级的Blocked，Suspend进程正好变为非阻塞状态，OS可以将执行进程转换为Ready,Suspend状态。

All—》Exit：通常，Running Exit。某些OS种，父进程可以终止子进程，使**任何状态的进程**都可转换为退出状态。

OS如何感知进程、控制进程及其所用的系统资源？



（实线表示**占用**，虚线表示**请求**）（主存可有多个进程））（操作系统就是公司财务）

**运行时系统管理结构：**

**系统中有大量关于进程和资源当前状态的信息**

**故每个进程和资源都要创建一个表来存放信息**

1. **内存表**

**进程的内存分配情况**

**进程的外存分配情况**

**访问共享内存区的保护属性（进程在内存中的操作权限）**

**管理虚拟存储需要的信息**

1. **IO表**

**空闲、已分配的IO设备**

**IO的状态**

**IO来源和目的地的内存地址**

1. **文件表**

**存在的文件**

**外存的文件**

**文件的当前状态**

**文件属性**

**有时文件信息由文件管理系统维护**

1. **进程表（链表，索引表，每创建一个进程就在主进程表创建一个进程表）**

**进程的位置**

**管理所需要进程的属性**

**-进程ID**

**-进程状态**

**-进程在内存的地址**

**-进程包含一组程序**

局部和全局变量的数据位置

任何定义的常量

堆栈信息（保护程序的当前状态）

**-进程控制块（PCB）**

所有进程的属性

**-进程映像**

进程的所有信息（程序、数据、堆栈、属性）的合集就是进程映像。

用户数据、用户程序、系统堆栈（存放系统及过程调用地址、参数），PCB（OS感知进程、控制进程的数据结构）

**PCB（Process Control Information）：是OS控制和管理进程时所用的基本数据结构。进程的PCB一定常驻内存的**

**作用：**PCB是相关进程存在于系统中的**唯一标志**；系统根据PCB而感知相关进程的存在。

PCB一定在内存，若在外存，则系统无法感知进程

**内容：**通常情况下，PCB包括的信息有进程的标识、处理机的状态、进程的控制信息、用户堆栈信息、私有的用户地址空间、共享空间指针等状态信息。

标识（ID）：

进程ID、父进程ID、用户ID（我用户是谁）

状态：

**用户可见寄存器**（用户直接访问可见的寄存器，通常8-32个寄 存器 ，和RISC精简指令集一般有100个以上的寄存器）

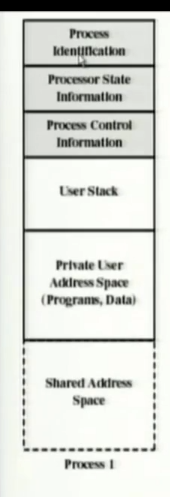
**控制和状态寄存器**（PC计数器：记载程序下一条记载到哪里； 条件代码：算术计算和逻辑计算的中间结果；状态信息：中断的信息）

堆栈（先进后出的堆栈）指针（存储过程或者系统调用的参数和地址，当调用这些信息、自己被调用的时候，就要把信息放到堆栈里面，之后再恢复回来。）

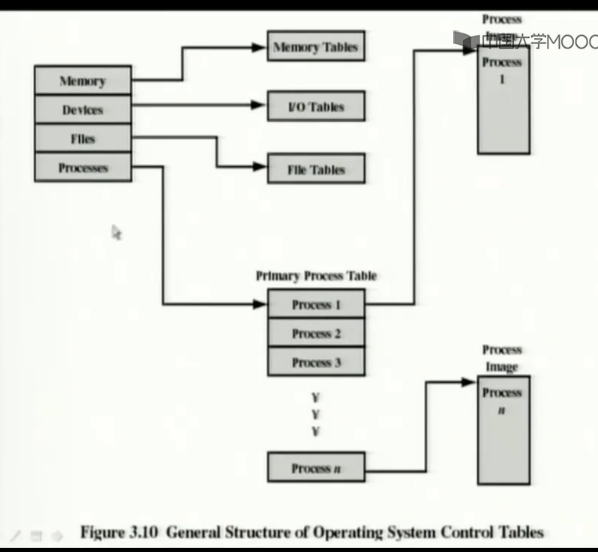
控制：

进程的当前状态、优先级、与调度相关的信息、事件

线程通信信息、内存管理信息、资源拥有和利用



**PCB包括的信息**有进程的标识、处理机的状态、进程的控制信息、用户堆栈信息、私有的用户地址空间、共享空间构成进程的逻辑抽象，进程映像。



**内核功能：资源管理功能和支撑功能（由原语支撑的）**

资源管理功能：（全由内核完成）

进程管理：进程创建、终止、调度、状态切换、同步和通信、管理PCB

内存管理：为进程分配地址空间、对换、段/页管理

IO管理：缓存管理、为进程分配IO通道和设备。

支撑功能:

中断处理

时钟管理

**原语**（系统提供进程创建终止，进程切换，阻塞与唤醒、挂起与激活这些功能以**原语的方式**提供出来的，原语与一般程序的区别就是不可中断）

统计

监测

**原语：特殊例程，不可中断**

**1、进程切换：（当进程被中断时，进行切换 ）**

**什么时候会切换：时间中断**（基于时间片的系统）**、IO中断、存储访问失效**（指虚拟存储，当要访问的数据不在内存就中断）**、陷阱**（error发生）**、管理程序调入**（例如打开文件）

**进程切换的过程状态：保存进程的信息、更新当前进程PCB中的信息、移动PCB到相应的队列、选择另外一个进程执行、更新被选择进程的PCB、更新内存管理数据结构**（就是新进程的内存空间地址如何分配）、恢复被选中进程的信息（PCB中信息读到相应的寄存器中）

**进程切换：是作用于进程之间的一种操作。**收回当前进程CPU并分配给另一就绪进程。

**##进程切换过程中一定有模式切换，一定要中断、执行调度程序，中断、调度程序是系统态的。**

**模式切换：用户态《——》系统态（为保护系统数据不被轻易篡改）**

**是进程内部所引用的一种操作。**进程中的程序调用核心子系统的系统调用。

**##模式切换不一定有进程切换，除非调用系统调用时，系统服务以进程的形式进行。**

1. **进程的创建和终止**

**创建：新的批作业提交、用户注册、提供服务（打印等）、进程创建进程（父子）。**

**创建过程：分配进程ID**（主进程表增加表项）**、分配空间**（地址空间、用户栈、PCB空间）**、初始化PCB**（状态、标识）**、建立链接**（若调度队列是链表，则插入就绪或（就绪、挂起（此时是内存不够放外存））链表中）、**建立或扩展其他数据结构**

**终止：** 遇到终止指令、用户注销、应用程序退出、error或fault错误、正常退出、越界、溢出、非法访问、企图执行特权指令

**终止过程：根据ID找到对应PCB读取状态、执行则终止并调度新进程执行、该进程子孙进程全终止、进程全部资源归还父进程或系统、从队列移除进程（僵死进程）PCB。**

**堵塞：**请求系统服务、启动某种操作、新数据未到达、无新工作可做

**（当出现阻塞事件，进程调用阻塞原语将自己阻塞，状态变阻塞、并进入相应的阻塞队列）**

**唤醒：事件发生、有关进程调用唤醒原语将阻塞进程唤醒、变为Ready进入 就绪队列。**

**挂起：出现挂起事件、系统将利用挂起原语将进程挂起，进程从内存换到外存。**

**激活：激活事件发生，系统利用激活原语将指定进程激活。进程从外存换入内存，并插入相应队列。**

**线程：（原因:进程过多，系统开销过大（进程管理））**

**线程也是程序的执行过程。**

中断时保存context，类似进程

执行堆栈（保存数据 ）

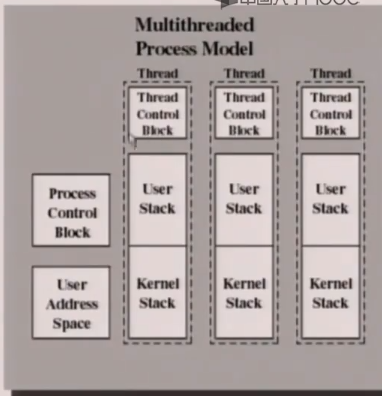
线程有少量的局部静态存储空间

线程访问其线程的资源和内存（线程共享进程资源）

**挂起一个进程会挂起其内所有线程（因为共享资源和内存）**

**##进程是资源分配单位、线程是调度单位（不用释放切换，多线程保证并发度的同时减少系统开销）**

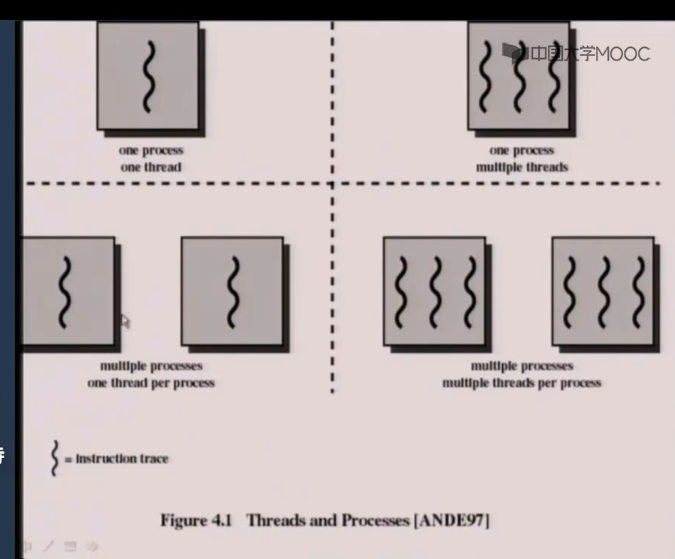
**##进程有PCB，线程有TCB（有用户栈、Kernel栈，没有用户地址空间（共享））**



##**线程优点：创建**线程时间更短、**终止**线程时间更短、同进程中线程**切换**时间更少、 可以互相沟通而不用进行模式切换。

Unix：多进程多线程

Windows、Linux：单进程多线程（只有一个初始进程：int，资源全由它分配，其下创建多线程）



单单、单多、多单、多多

线程状态：

**执行、就绪、阻塞**

状态转换：

派生、阻塞、不阻塞、完成

用户级线程：

线程管理全部由应用程序管理，内核不知道有这些线程的存在（TCB不一定有）。（核外子系统）

内核级线程：

进程及其线程由系统内核维护，调度基于线程本身（核心子系统）

组合级线程：Solaris、线程创建在用户态进行、调度同步在用户态（既有内核、也有用户线程）

**进程调度：**

调度类型，调度准则、调度算法、实时调度。

**调度目标：**

**响应时间**（评价分时系统）**、周转时间（**作业开始到完成时间、评价批处理系统**）、系统吞吐量、处理机效率、公平性**

**调度类型：**

**os划分：**

**批处理调度、分时调度、实时调度、多处理机调度**

**调度的层次划分：**

**长程调度：**外存到内存，基于作业

**决定多少、哪些程序创建进程到系统中**（创建进程）

**什么时候进行：**每个作业终止、处理器空闲一定时间段

**中程调度：外存与内存之间（就绪挂起）**

**目的：**是为了提高内存利用率和系统吞吐量

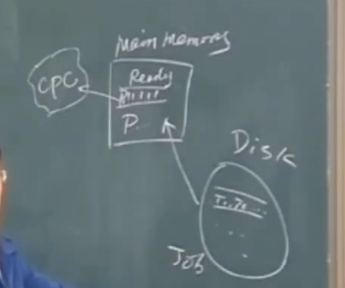
**短程调度：内存（）**

**调度内存中的就绪进程执行。**

决定就绪队列哪个进程获得处理机

#执行最频繁的

**什么时候执行？时钟中断、IO中断、系统调用、信号**



进程就绪，作业队列

面向用户目标：

**响应时间、周转时间、截止时间**（必须开始或结束、完成的时间、评价实时系统）

面向系统目标：

吞吐量（评价批处理系统）、处理机利用率、资源平衡（适用长程和中程调度）、公平性、优先级

**调度算法：**

**非剥夺方式：进程一旦执行，不可被中断，除非本身终止、IO阻塞**

**主要用于批处理系统**

**剥夺方式：进程不能长时间占用处理器，**主要用于实时性要求较高的实时系统以及性能要求较高的批处理系统和分时系统。

6种短程调度算法：

**FCFS（先来先服务算法）**：短进程（任务少，例如IO型进程）得等待很久，不公平，有利于计算性进程。一般与其他调度算法混合使用。

**轮转调度法**（针对分时系统）：设定每个进程的时间片

**虚拟轮转调度**（针对IO型进程）：由轮转调度算法改进，分两个就绪队列， 一个普通就绪，一个IO就绪队列（高优先级）。

短进程优先：

剩余时间短优先：

高响应比率优先：w+s/s

反馈调度法：

实时调度：（不再考虑响应时间和公平性）

实时系统中任务（task-structor）和进程同一个概念，

指及时响应外部事件的请求、在规定的时间内完成对该时间的处理，并控制所有实时任务协调一致地运行的计算机系统。

实时控制系统：主要用于生产过程的控制，实时采集数据并处理。

实时信息处理系统：指能对信息进行实时处理的系统。

周期性划分:周期性实时任务、非周期性实时任务（必须联系一个deadline）

截止时间：硬实时任务（必须满足任务对截止时间的要求）、软实时任务（）

特征：

确定性（确定的时间点或段执行，确定的中断操作接收）、响应性

特点：

快速上下文切换，任务小规模，能快速响应外部中断，多任务系统，文件存储速度快（无序存储数据，追加），基于优先级的调度。。。。。。。

实时调度：

基于时间片的轮转调度 秒级

基于优先级的非剥夺调度法 数百毫秒到数秒

基于优先级的剥夺调度法 几十毫秒或几毫米

立即剥夺调度法 （合适时间点才剥夺）

专门针对实时任务的调度：

静态表驱动调度法（根据表来调度下一个任务）

静态优先级调度法（依赖进程优先级）

动态计划调度法（依赖动态表来调度，用于周期性实时任务），最早截 止时间优先法

动态最大努力调度法（用于非周期实时任务调度），根据任务属性赋予优先级。

截止时间调度：

使用的信息：就绪时间、开始时间、完成时间、处理时间、资源请求、优先级。

一般基于最早截止时间优先

给了进程的开始截止时间，一般采用非剥夺的调度方式。

给了进程的完成截止时间，一般采用剥夺的调度方式。

最早截止时间优先:

周期型任务的完成截止时间：周期可预测，采用**静态表驱动之最早截止时间优先调度算法**。

**并发：**

互斥与同步、死锁和饥饿、临界区-临界资源（只能互斥使用）

进程通信、共享/竞争资源、多进程的同步问题，处理机时间分配

困难：共享全局资源不行、分配资源管理不行（要考虑更多因素）、程序错误 难以定位（可能因为多进程并发运行）

并发时全局变量必须控制。

并发系统运行时必须关注：

**持续跟踪活动进程的PCB**、**及时分配与回收资源**、**保护数据和资源**、**进程与其他并发的进程的执行次序必须无关**

进程的相互关系：（都会涉及互斥、死锁、同步的问题）

**进程间相互不知道、进程间接知道对方存在、进程直接知道对方存在**

**互斥：**

-临界区（**每个进程中访问临界资源的那段程序称为临界区**）：临界区一次只允许一个进程访问，临界资源只能互斥使用（例如 打印机）

进入区（能否进去，判断：1.有无进程正在使用该资源

2.无则进入并设置状态

）-临界区-退出区（退出通知），控制这一过程就是**互斥机制。**

##写数据一定互斥，保证数据完整数据（类似数据库的事务处理），防止脏数据。进程消息传递不要求互斥。进程等待信息可能导致死锁（消息丢失，互相等待消息）。多进程通信可能导致饥饿现象，其中两个特别活跃，第三个搭不上话。

**互斥要求：1、忙则等待**

**2、（让权等待）非临界区终止该进程的执行也不能影响其他 进程。（资源 归还，退出标识正常完成），互斥时不 允许死锁与饥饿**

**3、空闲让进**

**4、不能假设其他进程的数量和进度。**

**5、有限等待**

互斥实现方法:

**1.软件方法**：只能控制两个进程互斥，对多数量是不使用的。需要较 高的系统代价（需要额外资源）。有逻辑错误风险。

通过访问全局判定是否空闲，0则空闲，1则忙。

内存访问级，访问内存的同一区域由一仲裁程序控制。

**2.硬件方法**：（检测忙等消耗资源）

1. **屏蔽中断**（其他进程无法抢占资源，高代价，多处理器不 能保证互斥（不适合多处理机系统）

屏蔽中断-》进入临界区、退出-》打开中断

1. **专用机器指令**：一个机器指令内没有中断点、可以有DMA 断点。这样避免了资源冲突实现互斥。

Test-Set指令

Exchange指令

优点:可控制多进程的互斥使用。

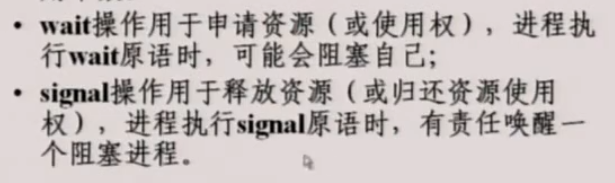
缺点:忙等、有可能饥饿、死锁

**3.信号量方法**：（特殊共享变量，两个原子操作）

**信号量：进程之间的通信方式，是有一个整型value的变量（可 能初始化为一个非负整数）。**

**如果进程在等待一个信号量，则阻塞自己。**

**Wait（s-1递减，申请资源或使用权且可能阻塞自己）和Signal（s+1释放资源或归还使用权并唤醒阻塞进程队列的第一个）是原语（接近机器指令），不能被中断。（s为信号量，有两个域，下见）**

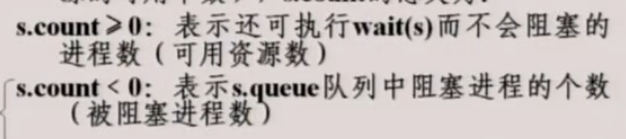


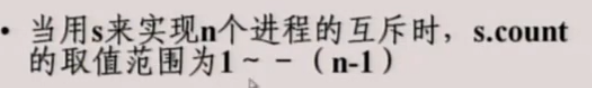
**通用信号量：记录型可为任何值。其中一个域为整形，另一 个域为队列（存放等待该信号量的阻塞进程 PCB，根据PCB修改状态）。**

**信号量类型：**

**互斥信号量：用于申请或释放资源的使用权，常初始化为1.**

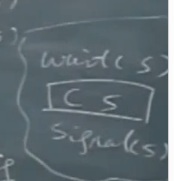
**资源信号量 ：用于申请或归还资源，可以初始化为大于1的 正整数，表示系统中某类资源的可用个数。**





**##申请到资源，还得得到使用该资源的使用权**

二进制信号量：只能是0-1



**4.管程（Monitor）方法：**（面向对象方法，包含数据、过程、初始化 语句）

**用信号量实现互斥，wait、signal的顺序和位置极其重要，编程容易出错。而管程实现互斥比用信号量实现互斥更简单、方便**

管程就是把临界资源的使用（wait、signal）**包装起来成一个类**，调用其中的**库函数**来调用资源。

#局部变量只能在管程里使用；进程使用临界资源通过调用管程中的一个过程实现；互斥，一次只允许一个进程在管程内运行。

**5.消息传递方法：（可用于互斥、交换信息）**

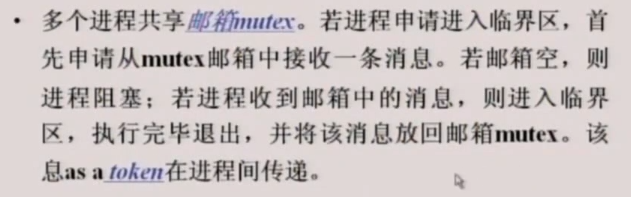
两个关键操作：send（destination、message）、receive（sorce，message）。（原语级别）

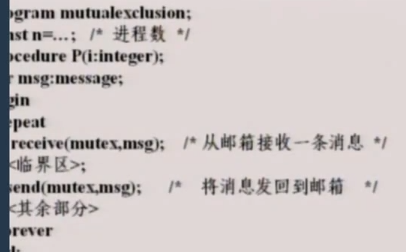
对于接收方、发送方可决定是否**阻塞发送、接收（是否等待）**

**1.若采用阻塞发送、接收方式，两者可能都阻塞等待（有无信息发送，接收；无则等待）。**

**2.若非阻塞发送、阻塞接收，发送方一直发送，接收方无则阻塞等待。**

**实现互斥：（取消息阻塞实现互斥，放回邮箱非阻塞）**





#设置两个邮箱同样机制，可实现生产者、消费者问题

**3.非阻塞发送、接收，两者均不等待。**

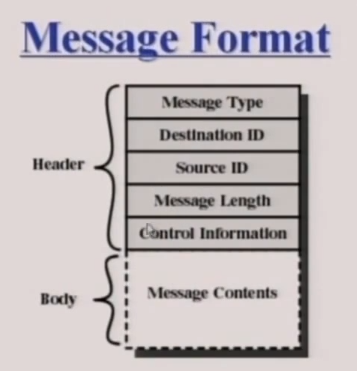
**寻址方式：**

**1.直接寻址：Sender、Receive都互相知道对方地址**

**2.间接寻址：消息发送到一个共享的数据结构（mailBoxs）， 发送方和接收方从该mailBoxs发取操作**

**#接收方只对一个进程，就是端口Port；接收方对应多个进程，就是mailBoxs。**

**消息是有格式的（Header、Body）**



（消息类型、目的地址、来源地址、消息长度、控制信息（例如校验信息）、消息主体）

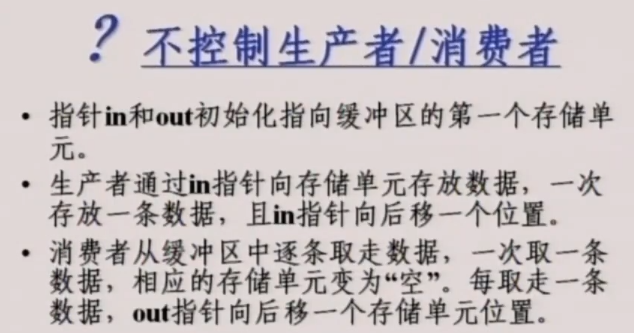
**生产者与消费者问题：**

生产者：生成数据、放置数据到一个缓冲区（放数据）

消费者：从缓冲区取数据（取数据）

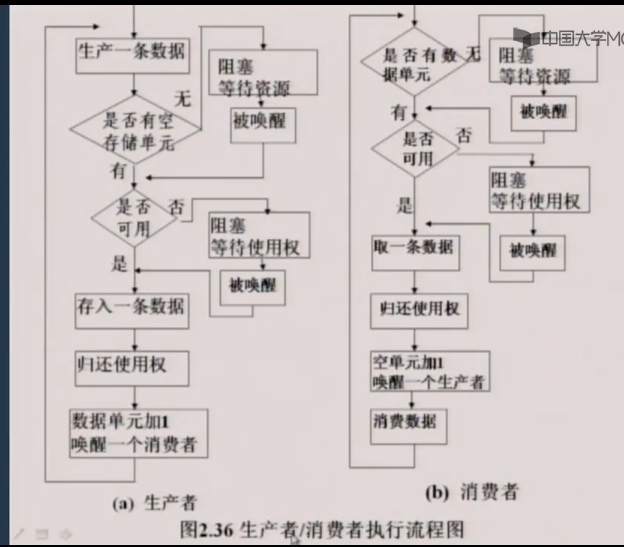
#一次只允许一个生产者或消费者进入缓冲区（互斥）

#生产者和消费者**同步**读/取buffer（不能满buffer写/空buffer读， 并wait，此时有互斥信号量，也有资源信号量）



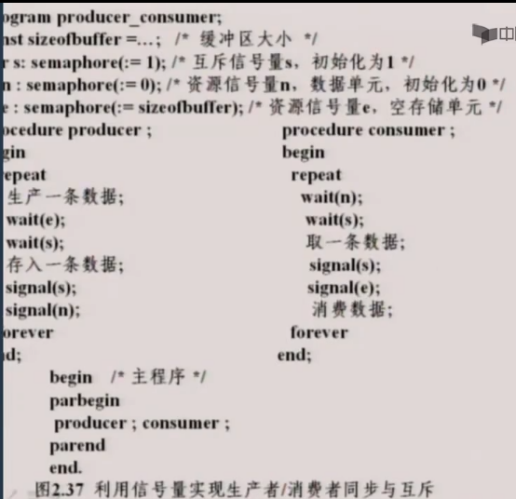
不控制的话：out、in不知道，会出现生产者、消费者同时进入缓冲 区，将导致脏数据。

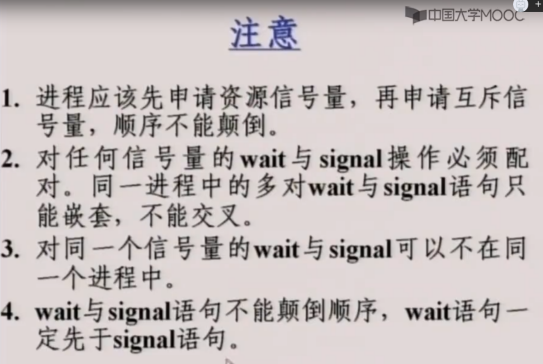
故生产者与消费者必须**互斥**。



**wait不能调换s，n（e）位置**，因为若先得到使用权，倘若此时资源队列为满，发现得不到资源就会等待，使用权不释放出来，其他进程就无法执行并释放资源，此时产生死锁。

**Signal不能调换s，n（e）位置，因为一旦调换，在生产者未归还使用权时，就有消费者唤醒进入缓冲区，造成脏数据，不能实现互斥。**



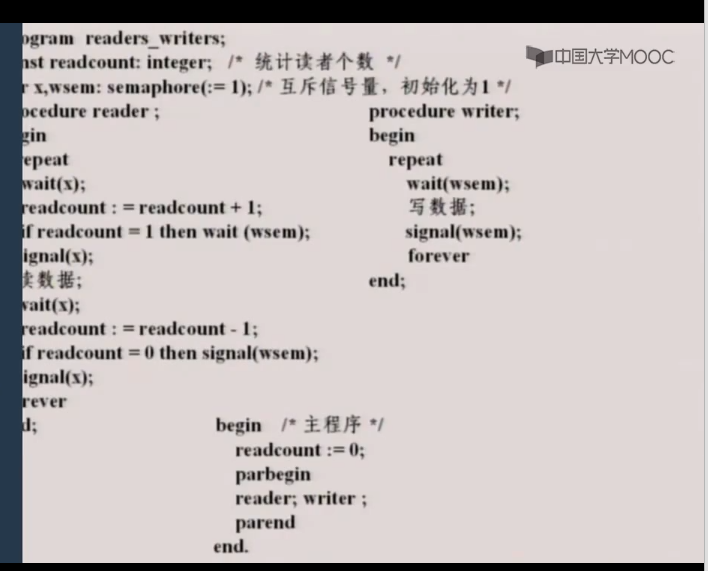


**读写者问题：**

**多个读者可同时读数据。但不希望写的时候读。且只能有一个读。**

**读优先：全读完，再写（简单，写者可能饥饿）**

**readcount变量是互斥变量，为了统计读进程的个数。（类似登记）**



**写优先：写者来了，不准读者进，等其内读者出完，写者进。（难）**

**死锁：**P1占A，P2占B；P1申请B，P2申请A。无法解除。

**死锁就是一组进程因为竞争资源或互相通信导致的永久阻塞。（目前无有效方法解决，只能尽量规避）**

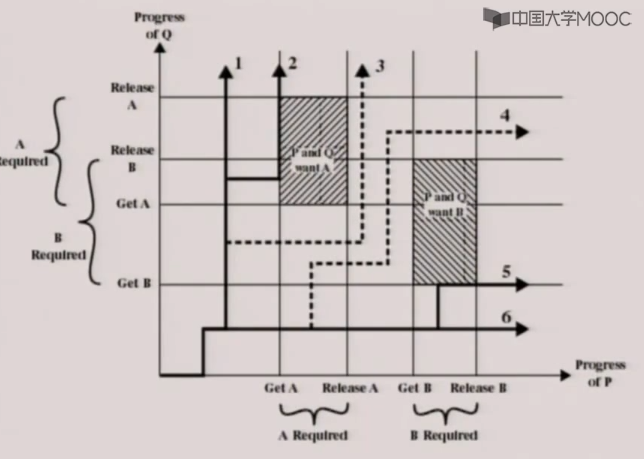
**#存在单进程死锁**

**#一旦死锁发生，只能由外力解除**

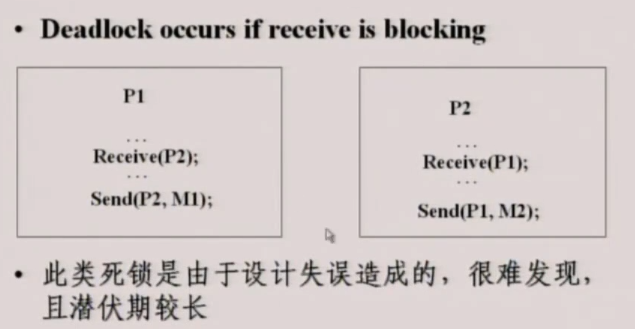
**原因：程序推进顺序不一样；竞争资源不当**



#可调换推进方式解决死锁（例如申请就用，用完再申请下一个资源），但是对于编写程序来说不切实际。



**可重复资源：**不会被消耗，用完返还，可能导致死锁（例如CPU，IO通道， 主存外存，文件数据库和信号量）



**可消耗资源：**由进程创建和摧毁（例如中断、信号量、消息、IO缓冲区信息）

**产生死锁的条件：**

**1.互斥**（可能，必要条件）

**2.保持并等待**（可能，必要条件）

**3.不剥夺**（可能，必要条件）

**4.环路等待**（一定有，充分条件，这是前面3个条件产生的结果）

**死锁的解决方法：**

1.**预防死锁（实施限制条件，降低性能）**

**间接方法**：禁止前3个条件之一

互斥不行，是系统的固有属性，**不能禁止**

保持并等待：要求进程一次性申请其所需全部资源，若系统 中没有足够资源分配则进程阻塞。**（低效）**

不剥夺：变成可剥夺。

（已占用同时申请资源，需先释放源，上面已举例） （根据优先级剥夺，但要求此类资源易于保存和恢 复，否则不能剥夺，例如打印机）**（条件苛刻）**

**直接方法：**禁止环路等待**（低效）**

将系统所有资源按类型不同线性排队，并赋予不同的序号。进程对某类资源的申请只能按照序号递增的方式进行。

**2.避免死锁（分配资源前，计算此次分配是否导致死锁，是则不分配直到 不会产生死锁才分配）（动态决策）（代价较高，常用于 服务器）**

**#必须申明每个进程所需资源总量；**

**#进程相互独立，执行顺序取绝系统安全而非进程间同步要求；**

**#系统必须提供固定资源供分配**

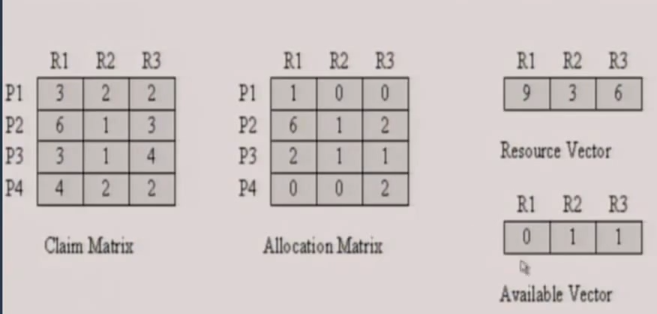
**#进程占有资源，则不能让其退出系统**

**银行家算法（需要检测进程分配到资源的状态）**

用于在银行发放贷款前，预测此次贷款是否引起资金 周转问题。

**1.安全状态：存在一个序列不会导致死锁则是安全状态**

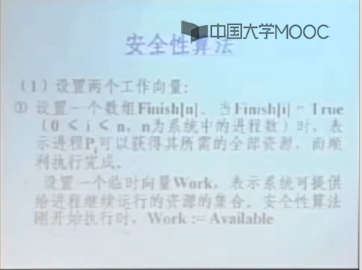
**2.不安全状态：不存在这样的序列，可能会导致死锁**

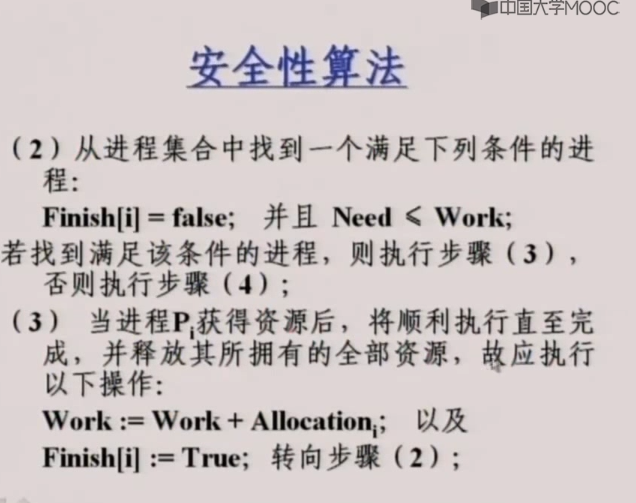


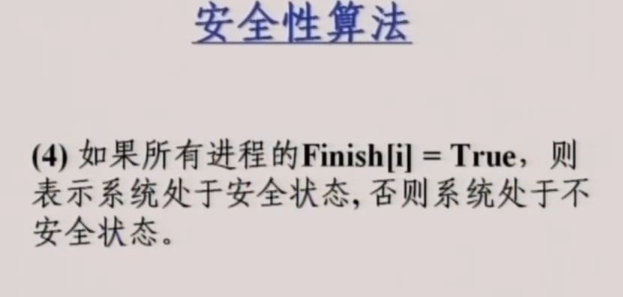
申请资源矩阵，分配资源矩阵，总资源矩阵，可用资源矩阵

**计算需求矩阵**：得出Needm=CM--AC（申请-分配），根据可用 矩阵分配。（确保分配是安全的，能释放资源）

**#若试探分配不安全，就让该进程阻塞等待。**







**3.检测死锁**

**不同于预防死锁，不限制资源的访问方式和资源申请；os周期性检测死锁（是否出现环路等待）**

**（1）死锁检测策略：**

**1.若检测到死锁，中断所有进入死锁的进程（后面可能还会发生）**

**2.若检测到死锁，返回至未出现死锁前的检测点（还会出现）**

**3.若检测到死锁，逐步剥夺死锁进程资源直至死锁解除**

**（2）选择哪些进程执行策略：**

**#选择现在消耗处理机资源最少的进程**

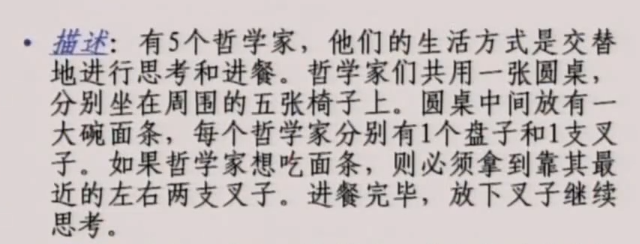
**#选择现在产出较小的进程**

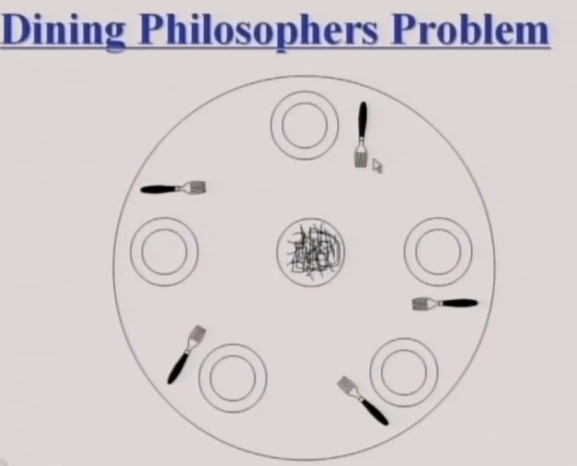
**#剩余时间最长的进程**

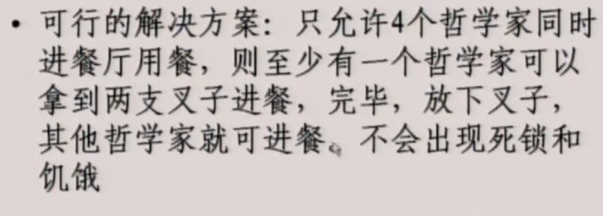
**#执行时间最短的进程**

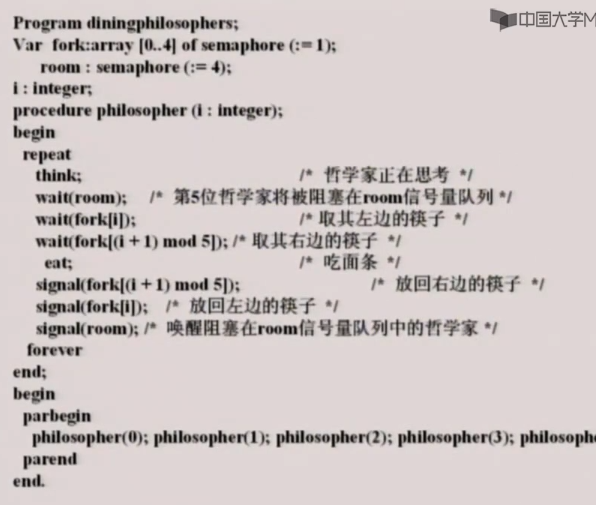
**#低优先级的**

**哲学家进餐问题**









（能完全解决死锁）