第一章：计算机系统概述

一、指令的执行

1. 指令寄存器（IR）：用来暂存指令，当CPU执行指令时，先把它从内存取到缓冲寄存器中，再送入IR。然后指令译码器从IR中将指令取出来，进而分析指令。

2. 程序计数器（PC）：记住它保存的总是将要执行的下一条指令的地址即可。

2. 地址寄存器（AR）：保存当前CPU所访问的内存单元地址。由于内存和CPU在操作速度上的差异，所以需要用AR保存地址信息，知道内存的读写操作完成为止。

4. 指令译码器（ID）：用来分析指令。指令包括操作码和地址码两部分，指令译码器就是对操作码字段进行分析解释，识别该指令规定的操作，向操作控制器发出控制信号，完成所需功能。

指令的执行过程：

> 指令的执行需要完成取指令、分析指令和执行指令的操作，过程分为：取指令、指令译码、按指令操作码执行、形成下一条指令地址等。
>
> （1）在程序执行之前，会将程序的起始地址送入程序计数器中，该地址在程序加载到内存时确定，所以程序计数器中首先存的是程序第一条指令的地址。将该地址送往地址总线，完成取指操作。
>
> （2）取来的指令暂存到指令寄存器中。
>
> （3）指令译码器从指令寄存器中得到指令，分析指令的操作码和地址码。然后 CPU 根据分析的操作码知道该条指令要进行的操作，根据地址码找到需要的数据，完成指令的执行。
>
> （4）程序计数器加1或根据转移指令得到下一条指令的地址，接下来再进行下一条指令的执行，直到整个程序执行完成。

二、中断和指令周期

指令周期：CPU从主存中取出并执行一条指令的时间,包括取指、间址、执行(取操作数)、中断四部分。

指令周期包括若干机器周期，一个机器周期包括若干时钟周期。

发生了中断，就意味着需要操作系统介入，开展管理工作。由于操作系统的管理工作需要使用特权指令，因此CPU要从用户态转核心态。**中断**可以使CPU从**用户态切换为核心态，使操作系统获得计算机的控制权。**有了中断，才能实现多道程序并发执行。

内中断（异常、例外、陷入）：CPU内部，**与当前执行的指令**有关。

外中断：信号来源**CPU外部**，与当前执行的指令**无关。**

> 用户态、核心态之间的切换使怎么实现的？
>
> 答：“**用户态 -->核心态**”是通过**中断**实现的。并且**中断是唯一**途径。“**核心态 -->用户态**”的切换是通过**执行一个特权指令**，将程序状态字的标志为设置为“用户态”。

中断不可预测，可嵌套，在中断处理过程中，处理器可把响应此中断的**中断处理程序入口地址**装入PC。有顺序处理和嵌套处理两种方法。

第二章：操作系统概述

一、三种重要接口

典型计算机系统中的三种重要接口:

> 指令系统体系结构（ISA）：定义了计算机遵循的机器语言指令系统，该接口是硬件与软件的分界线。注意，应用程序和公用程序都可直接访问ISA，这些程序使用ISA的一个子集（用户级ISA）。操作系统能使用其他一些处理系统资源的机器语言指令（系统级ISA）
>
> 应用程序二进制接口（ABI）：定义了程序间二进制可移植性的标准。
>
> 应用程序编程接口（API）：允许应用程序访问系统的硬件资源和服务。这些服务用户级ISA和高级语言库（HLL）调用来提供

二、简单批处理

监控程序功能:

- 作业的自动续接
- 内存保护：保护监控程序所在的内存区域
- 定时器：防止某作业独占系统
- 特权指令：只可由监控系统执行的指令
- 中断

> 正是内存保护和特权指令的出现引入了运行模式，不同模式的访问权限不同
>
> - **用户模式：用户程序以此模式执行，有些内存区域受到保护，特权指令不允许执行**
> - **内核模式：监控程序以此模式执行，可以执行特权指令**

三、多道批处理

多道程序设计可以现主提高系统设备利用率:

- 内存中存放多个作业
- 多个作业可并发执行
- 作业调度程序

硬件支持：支持I/O中断和DMA的硬件。

第三、四章：进程和线程

一、进程的描述和控制

进程的定义:

> **进程**是指**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**。
>
> **进程实体（进程映像、进程）**：程序段、相关的数据段、进程控制块PCB。

进程的特征:

> 结构特征：由程序段、数据段、进程控制块三部分组成；
> 动态性：进程的实质是程序的执行过程；动态性是进程最基本的特征。进程实体具有一定的生命周期。
> 并发性：多个进程可同存于内存中，能在一段时间内同时运行；引入进程的目的正是为了使

> 进程实体能够并发执行； 程序不能参与并发执行。
>
> 独立性：进程是独立运行的基本单位，独立获得资源和调度的基本单位；
>
> 异步性：各进程按各自独立的不可预知的速度向前推进。

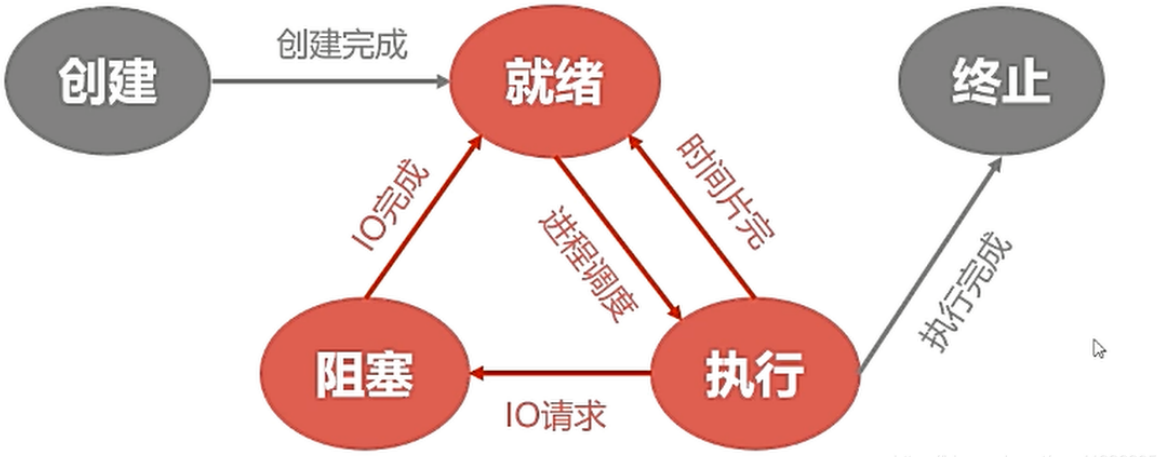
进程控制块:

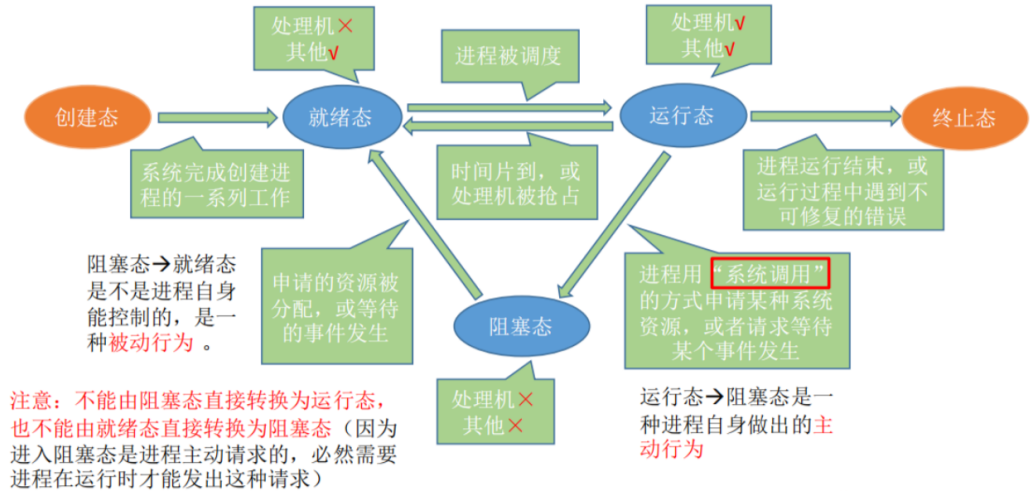
> 为了描述和控制进程运行，系统为每个进程定义了 一个数据结构，**称为进程控制块（PCB）。 PCB中记录了OS所需的、用于描述进程当前情况以及控制进程运行的全部信息。*PCB是操作系统中最重要的记录型数据结构。***

进程的三种基本状态:

> 就绪状态：**当进程已经获得了除CPU以外的所有资源， 一旦得到CPU，就立即可以运行，则这些进程所处的状态为**就绪状态。
>
> **运行状态：**正在运行的进程所处的状态为**运行状态。**
>
> **阻塞/等待状态：**若一进程正在等待某一事件发 生（如等待输入/输出工作完成），这时即使给它CPU，它也无法运行，称该进程处于等待状态，或**阻塞、睡眠、封锁状态。 阻塞队列：**系统根据阻塞原因设置多个队列来管理处于阻塞状态的进程。

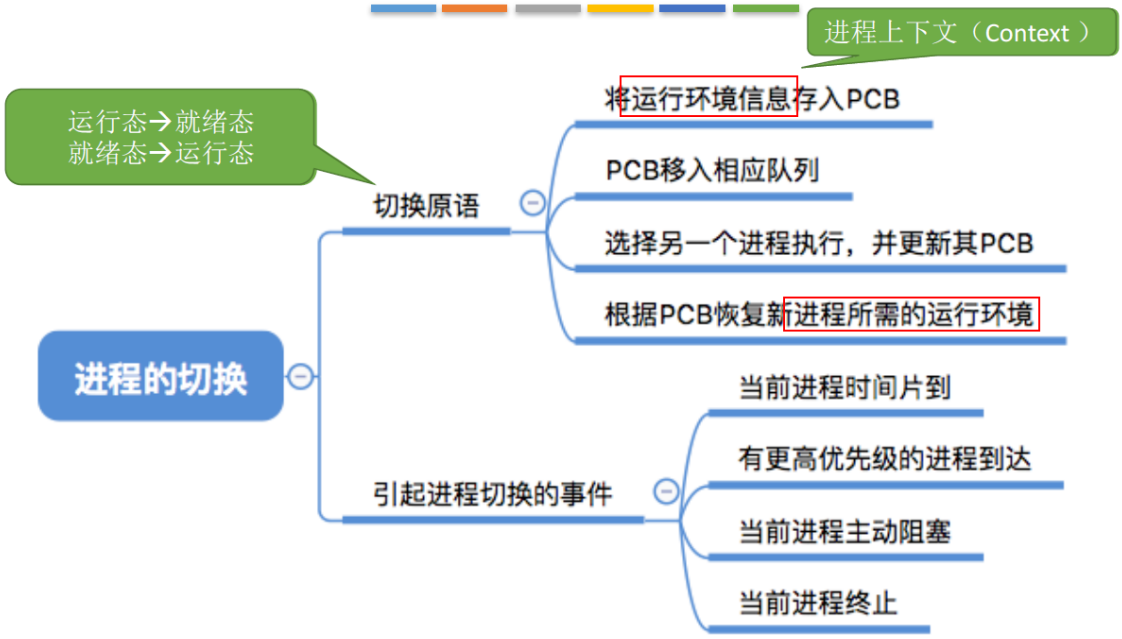
五状态进程模型:





1、就绪态不能转换为阻塞态：因为**阻塞态**是处于**运行态**的进程在运行时主动**执行造成阻塞的代码**而导致的：

2、阻塞态不能转换为运行态：只有被**调度**的进程才会转入运行态，而只有处于就绪态的进程才会被调度，因此阻塞态必须经过就绪态后才能转换为运行态。



> 模式切换可在不改变运行态进程状态（有一些中断 / 异常不会引起进程状态转换，不会引起进程切换，只是在处理完成后把控制权交还给被中断进程。）的情况下发生，此时保存上下文并在以后恢复上下文需要的开销很少。但是若当前正运行进程将转换为另一种状态（就绪（如时钟中断）、阻塞（如系统调用）等），则操作系统必须让环境产生实质性的变化。
>
> **进程切换**必须在操作系统的内核模式下进行。

二、线程

进程和线程：

> 进程（Process）是资源分配的基本单位，线程（Thread）是CPU调度的基本单位。
>
> 线程将进程的资源分和CPU调度分离开来。 以前进程既是资源分配又是CPU调度的基本单位，后来为了更好的利用高性能的CPU，将资源分配和CPU调度分开。因此，出现了线程。
>
> 进程和线程的联系： 一个线程只能属于一个进程，一个进程可以拥有多个线程。线程之间共享进程资源。
>
> 进程和线程的实例：打开一个QQ，向朋友A发文字消息是一个线程，向朋友B发语音是一个线程，查看QQ空间是一个线程。QQ软件的运行是一个进程，软件中支持不同的操作，需要由线程去完成这些不同的任务。

线程分类:

> 用户级线程:由应用程序通过线程库实现。所有的线程管理工作都由应用程序负责（包括线程切换)
> 用户级线程中，线程切换可以在用户态下即可完成，无需操作系统干预。
> 在用户看来，是有多个线程。但是在操作系统内核看来，并意识不到线程的存在。（用户级线程对用户不透明，对操作系统透明)
> 可以这样理解，“用户级线程”就是“从用户视角看能看到的线程”。
>
> 内核级线程:的管理工作由操作系统内核完成。线程调度、切换等工作都由内核负责，因此内核级线程的切换必然需要在核心态下才能完成。
> 可以这样理解，“内核级线程”就是“从操作系统内核视角看能看到的线程”。
> 操作系统只“看得见”内核级线程，因此只有内核级线程才是处理机分配的单位。

	要做什么	调度发生在..	发生频率	对进程状态的影响
高级调度（作业调度）	按照某种规则，从后备队列中选择合适的作业将其调入内存，并为其创建进程	外存→内存（面向作业）	最低	无→创建态→就绪态
中级调度（内存调度）	按照某种规则，从挂起队列中选择合适的进程将其数据调回内存	外存→内存（面向进程）	中等	挂起态→就绪态（阻塞挂起→阻塞态）
低级调度（进程调度）	按照某种规则，从就绪队列中选择一个进程为其分配处理机	内存→CPU	最高	就绪态→运行态

第五、六章：并发

一、信号量

临界资源：多个进程可以共享系统中的各种资源，许多资源一次只能为一个进程所用，将一次仅允许一个进程所用的资源称为临界资源。

临界区：每个进程中访问临界资源的那段代码。

信号量用来解决互斥和同步问题，它只能被两个标准的 原语 wait(W)和signal(S)访问，也可以记为 P操作 和 V操作。

原语：指完成某种功能且不被分割、不被中断执行的操作序列，通常可由硬件来实现。

互斥：上P下V围堵，初值为1。　　同步：前V后P相接，初值为0。

生产者消费者：

```
semaphore mutex = 1;
semaphore full = 0;
semaphore empty = n;

producer()
{
    while (1)
    {
        produce an item in nextp;
        P(empty);
        P(mutex);
        add nextp to buffer;
        V(mutex);
        V(full);
    }
}

consumer()
{
    while (1)
    {
        P(full);
        P(mutex);
        remove an item from buffer;
        V(mutex);
        V(empty);
```

```
        consume the item;
    }
}
```

若生产者进程先P(mutex)，然后执行P(empty)，消费者先执行P(mutex)，然后执行P(full)，可不可以?

> 不行，假设缓冲区已满，先运行生产者进程，它先互斥加锁，然后执行P(empty)被阻塞，这是去运行消费者进程，执行P(mutex)，但此时锁在生产者进程，因此消费者进程也被阻塞，导致无休止的等待。

二、死锁

死锁定义：如果一组进程中的每一个进程都在等待由该进程中的其他进程才能引发的事件，那么该组进程是死锁的。

饥饿定义：指系统不能保证某个进程的等待时间上界，从而使该进程长时间等待，当等待时间给进程推进和响应带来明显影响时，称发生了进程饥饿。当饥饿到一定程度的进程所赋予的任务即使完成也不再具有实际意义时称该进程被饿死。

死锁原因：**源于多个程序对资源的争夺，不仅对不可抢占资源进行争夺时会引起死锁，而且对可消耗资源进行争夺时**，也会引起死锁。

饥饿原因：**如果一个线程因为处理器时间全部被其他线程抢走而得不到处理器运行时间**，这种状态被称之为饥饿，一般是由高优先级线程吞噬所有的低优先级线程的处理器时间引起的。

死锁必要条件：**互斥条件，不可剥夺条件，请求和保持条件，循环等待条件。**

饥饿必要条件：没有其产生的必要条件，随机性很强。并且饥饿可以被消除，因此也将忙等待时发生的饥饿称为活锁。

死锁的处理策略：

- 预防死锁。破坏死锁产生的四个必要条件中的一个或几个。
- 避免死锁。用某种方法防治系统进入不安全状态，从而避免死锁（银行家算法，破坏循环等待条件）。
- 死锁的检测和接触。允许死锁的发生，不过操作系统会负责检测出死锁的发生，然后采取某种措施解除死锁。

安全序列、不安全状态、死锁的联系：

> 如果分配了资源之后，系统中找不出任何一个安全序列，系统就进入了不安全状态。这就意味着之后可能所有进程都无法顺利地执行下去。当然，如果有进程提前归还了一些资源，那系统也有可能重新回到安全状态，不过我们在分配资源之前总是要考虑到最坏情况。
>
> 如果系统处于安全状态，就一定不会发生死锁。如果系统进入不安全状态，就可能发生死锁（处于不安全状态未必就是发生了死锁，但发生死锁时一定是在不安全状态）。
>
> 因此可以在资源分配之前预先判断这次分配是否会导致系统进入不安全状态，以此决定是否答应资源分配请求。这也是“银行家算法”的核心思想。

第七、八章：内存

一、内存管理

创建进程**步骤**：编译（源代码→目标模块）、链接（目标模块+库函数→装入模块）、装入（装入模块→内存）

内部碎片：程序小于固定分区大小，也占用一个完整的内存分区空间。

外部碎片：所有分区外的存储空间会变成越来越多的碎片。

逻辑地址空间：链接程序顺序依次按各个模块的相对地址构成统一的从0号单元开始编址的逻辑地址空间。

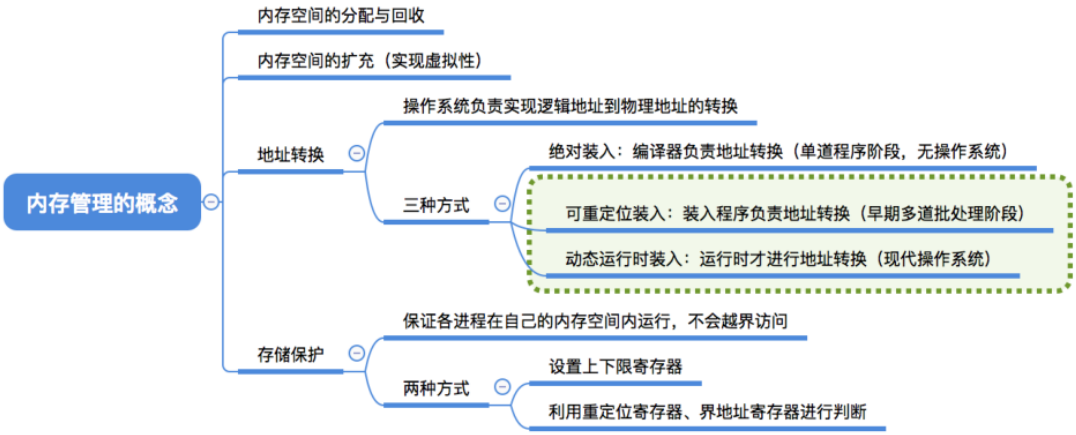
物理地址空间：内存中物理单元的集合。

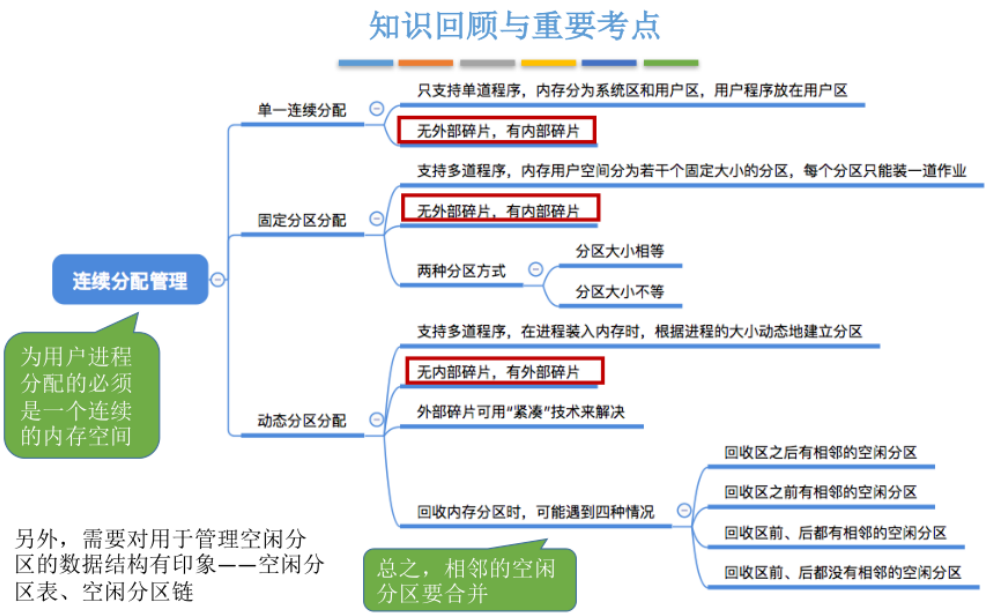
地址重定位：通过地址转换将逻辑地址转换为物理地址。

知识回顾与重要考点

算法	算法思想	分区排列顺序	优点	缺点
首次适应	从头到尾找适合的分区	空闲分区以地址递增次序排列	综合看性能最好。***算法开销小***，回收分区后一般不需要对空闲分区队列重新排序	
最佳适应	优先使用更小的分区，以保留更多大分区	空闲分区以容量递增次序排列	会有更多的大分区被保留下来，更能满足大进程需求	会产生很多太小的、难以利用的碎片；***算法开销大***，回收分区后可能需要对空闲分区队列重新排序
最坏适应	优先使用更大的分区，以防止产生太小的不可用的碎片	空闲分区以容量递减次序排列	可以减少难以利用的小碎片	大分区容易被用完，不利于大进程；***算法开销大***（原因同上）
邻近适应	由首次适应演变而来，每次从上次查找结束位置开始查找	空闲分区以地址递增次序排列（可排列成循环链表）	不用每次都从低地址的小分区开始检索。***算法开销小***（原因同首次适应算法）	会使高地址的大分区也被用完

知识回顾与重要考点





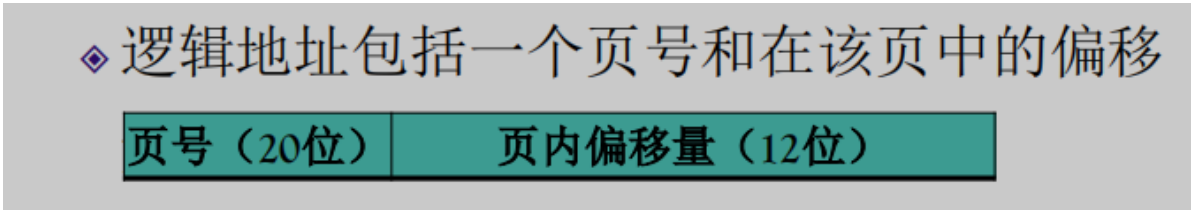
页表、页、页框：

分页存储管理是将一个进程的逻辑地址空间划分为若干个大小相等的片，这些片称之为页面，并编号第0页，第1页。。。同时，我们还把内存空间也划分为与页面大小相同的若干个存储快，称为块或叶框，也进行编号0#，1#...之后我们为进程分配内存时，即是将进程的若干个页分别映射装入到可以不相邻的块中去。由于这里进程的最后一页往往装不满块而会形成不可利用的碎片，我们称之为内部碎片。

页面大小：指一个页面占多大的存储空间。（一般为2的12次方，也就是4KB，详细内容可以去了解分页地址中的地址结构）。

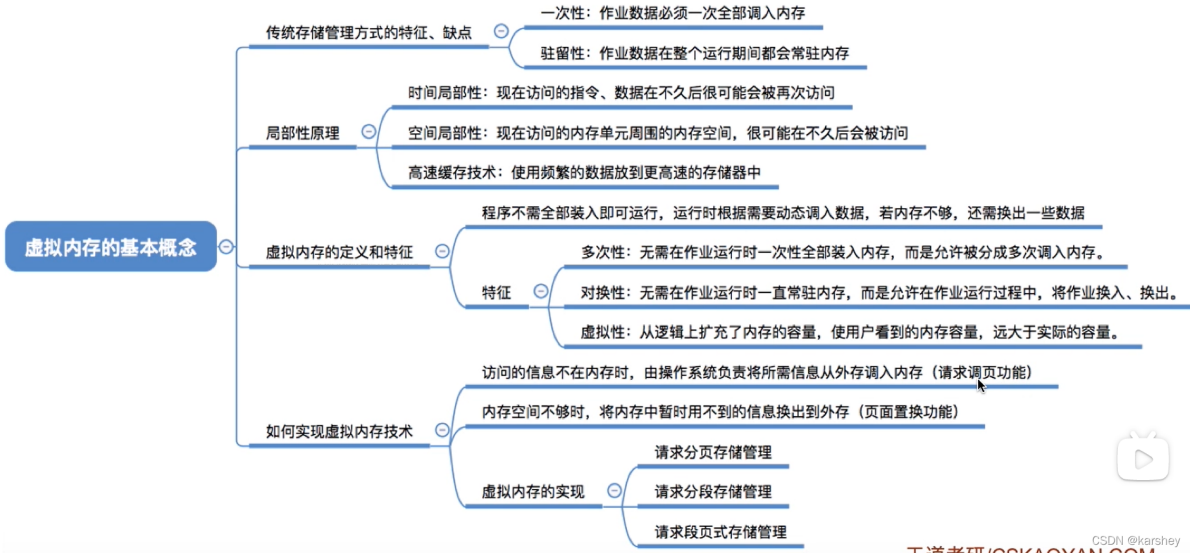
内存大小 = 页面长度×页面大小（物理块大小）

逻辑地址（页号，偏移量）（逻辑地址就是虚拟地址）



物理地址 = 页框号 * 页（框）的大小 + 偏移量。

二、虚拟内存



虚拟内存是一种内存管理技术，是虚拟的、逻辑上存在的存储空间。

页表项有：
|页号|物理块号|状态位P|访问字段A|修改位M|外存地址|

各个位的作用：
1）状态位P:用于指示该页是否已调入内存，供程序访问时参考；
2）访问字段A:用于记录本页在一段时间内被访问次数，供选择换出页面时参考；
3）修改位M：表示该页在调入内存后是否被修改过，供置换页面时参考；
4）外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

TLB的作用：

> 引入TLB前：CPU收到来自程序的虚拟内存地址后，首先需要去物理内存中取页表，然后对应程序传来的虚拟页面号，在页表中找到对应的物理页面号，然后才能访问实际的物理内存地址，整个流程中CPU至少访问两次物理内存，实际上可能更多次。因此为了减少CPU访问物理内存的次数，引入TLB。
>
> 引入TLB后：CPU收到来自程序的虚拟内存地址后，优先在TLB中进行寻址。TLB负责将虚拟内存地址转换为实际的物理内存地址，然后再去访问实际的物理内存地址。

在请求分页系统中，可以通过查询页表中的状态位来确定所要访问的页面是否存在于内存中。每当所要访问的页面不在内存时，会产生一次缺页中断，此时操作系统会根据页表中的外存地址在外存中找到所缺的一页，将其调入内存。

缺页次数指的是操作系统将页从外存调入内存的次数，而缺页中断次数指的是由于内存块数量的限制，将内存中暂时用不到的页面与外存中需要调入内存的页面交换的次数。

置换策略：OPT，FIFO，LRU。

第九章：调度

- 一个进程是一个程序对某个数据集的执行过程，是资源分配的基本单位。
- 作业是用户是要求计算机完成的某项任务所做工作的集合。（用在批处理系统，我的理解是批处理的bat文件，作业包括了至少一个进程，而且作业调度主要关心是外存调入内存的过程，进程是作业的执行过程）

所谓进程调度，就是**从进程的就绪队列（阻塞）中按照一定的算法选择一个进程并将 CPU 分配给它运行。**

	要做什么	调度发生在..	发生频率	对进程状态的影响
高级调度（作业调度）	按照某种规则，从后备队列中选择合适的作业将其调入内存，并为其创建进程	外存→内存（面向作业）	最低	无→创建态→就绪态
中级调度（内存调度）	按照某种规则，从挂起队列中选择合适的进程将其数据调回内存	外存→内存（面向进程）	中等	挂起态→就绪态（阻塞挂起→阻塞态）
低级调度（进程调度）	按照某种规则，从就绪队列中选择一个进程为其分配处理机	内存→CPU	最高	就绪态→运行态

周转时间

单个作业周转时间 = 作业完成时间-作业提交时间。

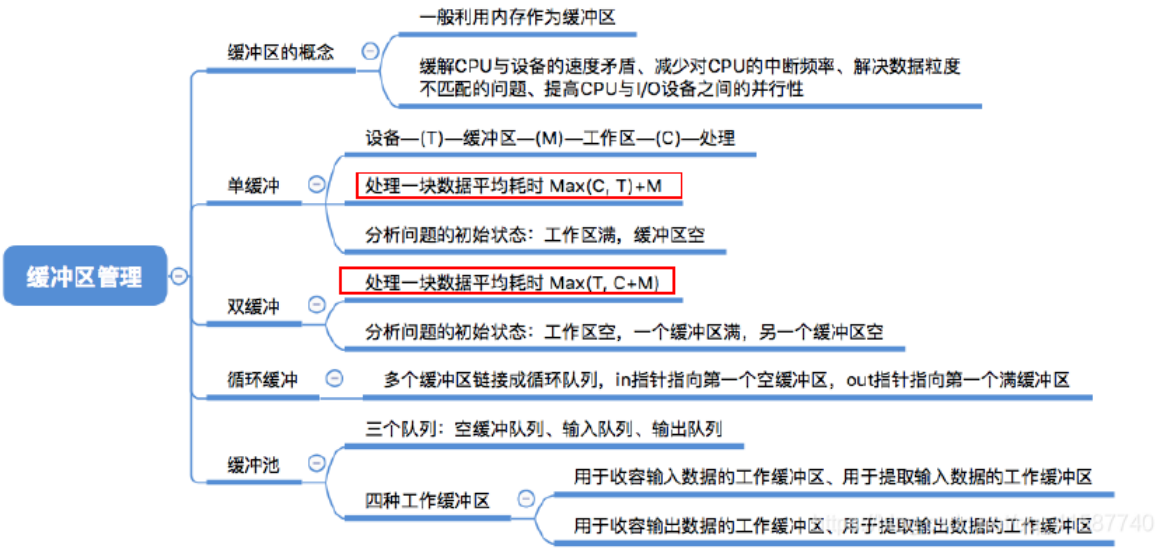
平均周转时间=各作业周转时间之和/作业数。

进程调度算法：

算法名称	算法思想	适用调度场景	是否可抢占	优缺点	是否导致饥饿
先来先服务算法（FCFS）	按照作业/进程到达的先后次序来进行调度	作业调度&进程调度	不可抢占	优点：简单、算法简单；缺点：作业的平均带权周转时间太长，对于短作业不友好	不会
短作业/进程优先算法（SJF/SPF）	要求最短的作业/进程先得到调度	作业调度&进程调度	通常不可抢占，但是有可抢占版本：最短剩余时间优先算法（SRTN）	平均周转时间通常是最短的	会，如果一直进入短作业，可能长作业会被饿死
高响应比优先算法（HRRN）	每次调度前就算每个作业/进程的响应比，响应比高的优先得到调度	作业调度&进程调度	非抢占式算法，当前作业/进程主动放弃cpu使用才需调度	综合考虑了等待时间和服务时间，对于长作业来说，随着等待时间增长，响应比也会增长，不会出现饥饿现象	不会

时间片轮转，抢占式。

第十一章：I/O管理和磁盘调度



平均磁盘访问时间 = 平均寻道时间 + 平均旋转延时 + 传输时间 （+ 控制器延时）。

磁盘调度策略：

> FIFO：最简单的调度，按照先进先出顺序处理队列中的项目。具有公平的特点，因为每个请求都会得到处理，并且是按照接收到的顺序进行处理的。
>
> SSTF：选择使磁头臂从当前位置开始移动最少的磁盘I/O请求。
>
> SCAN：要求磁头臂仅沿一个方向移动，并在途中满足所有未完成的请求，直到它到达这个方向上的最后一个磁道，或者在这个方向上没有别的请求为止。

第十二章：文件管理

文件组织指文件的逻辑结构，有以下5种：

堆

索引文件

顺序文件

索引顺序文件

直接或散列文件