第一章 概述

① 并发、并行

单核CPU只能并发，多核CPU才能并行

宏观上同时发生
微观上交替进行

微观上、宏观上均同时发生

共享：同时共享、互斥共享

- 同时共享方式：系统中的某些资源，允许一个时间段内由多个进程“同时”对它们进行访问。
- 互斥共享方式：系统中的某些资源，虽然可以提供给多个进程使用，但一个时间段内只允许一个进程访问该资源。

异步：

异步是指，在多道程序环境下，允许多个程序并发执行，但由于资源有限，进程的执行不是一贯到底的，而是走走停停，以不可预知的速度向前推进，这就是进程的异步性。

② 单道批处理系统：串行、总是等待IO执行

多道批处理系统：没有人机交互

分时操作系统：不能区分任务的紧急性

实时操作系统

③ CPU的内核态和用户态

访管指令：通过中断由硬件实现转换

处于内核态时，说明此时正在运行的是内核程序，此时可以执行特权指令
处于用户态时，说明此时正在运行的是应用程序，此时只能执行非特权指令

通过CPU的程序状态字寄存器（PSW）判断

仅指令（系统调用）可在用户态调用，但只能在内核态执行

I/O操作、设备中断，需在内核态

④ 中断的类型

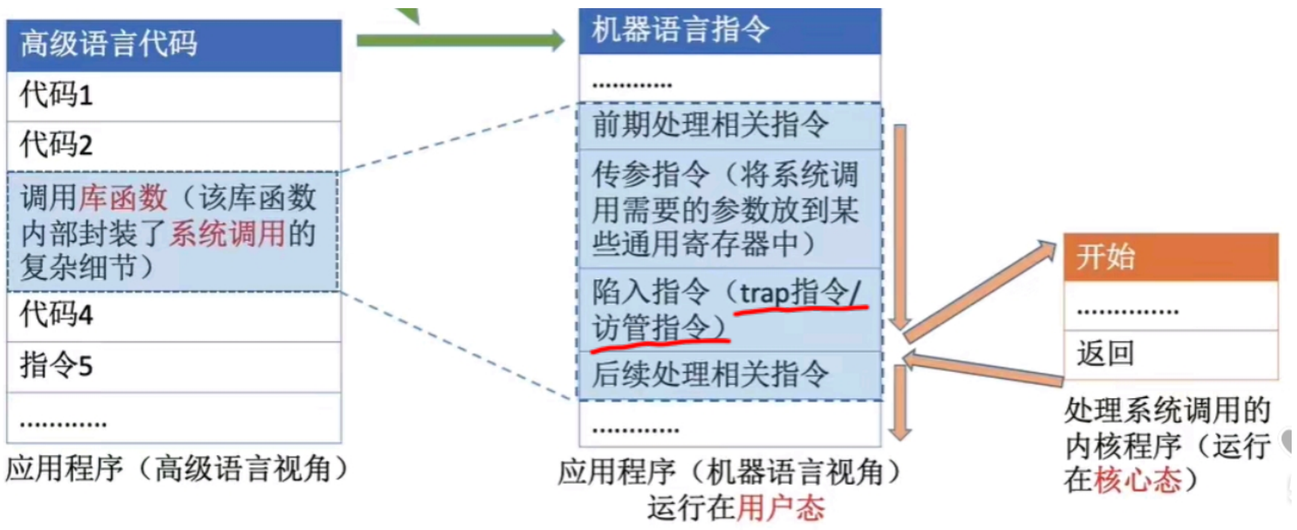
中断：也称外中断，是指来自CPU外部的事件。很典型的是一个时钟中断（并发运行的基础），表示一个固定的时间片已到，让处理机处理计时、启动定时运行的任务等。

异常：也称内中断，是指来自CPU内部的事件。如程序的非法操作码、地址越界、运算溢出等，异常不能被屏蔽，一旦出现就应该立即处理。

中断可以被屏蔽，而异常不可以

保存中断断点及状态是由硬件完成的

⑤ 系统调用



用户态发起指令　　内核态执行

Trap/访管指令在用户态执行，只是会引发内中断使CPU进入内核态

⑥ 操作系统的结构

	优点	缺点
分层结构	易于调试和验证 易扩充和易维护	仅可调用相邻层 效率低，不可跨层调用
模块化	易于维护，逻辑清晰 各模块之间直接调用	难以调式和验证
大内核	性能高，内核个功能直接调用	复杂，难以维护，一个功能坏全部瘫痪（可靠性低）
微内核	功能少，易于维护 某个出错不会导致整个崩溃	性能低，需要频繁切换状态
外核	更灵活使用硬件资源 减少资源的映射层提升效率	使系统更加复杂 降低系统的一致性

两类虚拟机管理程序

	第一类VMM	第二类VMM
对物理资源的控制权	直接运行在硬件之上，能直接控制和分配物理资源	运行在Host OS之上，依赖于Host OS为其分配物理资源
资源分配方式	在安装Guest OS时，VMM要在原本的硬盘上自行分配存储空间，类似于"外核"的分配方式，分配未经抽象的物理硬件	GuestOS 拥有自己的虚拟磁盘，该盘实际上是Host OS 文件系统中的一个大文件。GuestOS分配到的内存是虚拟内存
性能	性能更好	性能更差，需要HostOS作为"中介"
可支持的虚拟机数量	更多，不需要和 Host OS 竞争资源，相同的硬件资源可以支持更多的虚拟机	更少，Host OS 本身需要使用物理资源，Host OS 上运行的其他进程也需要物理资源
虚拟机的可迁移性	更差	更好，只需导出虚拟机镜像文件即可迁移到另一台 HostOS 上，商业化应用更广泛
运行模式	第一类VMM运行在最高特权级（Ring 0），可以执行最高特权的指令。	第二类VMM部分运行在用户态、部分运行在内核态。GuestOS 发出的系统调用会被 VMM 截获，并转化为 VMM 对 HostOS 的系统调用

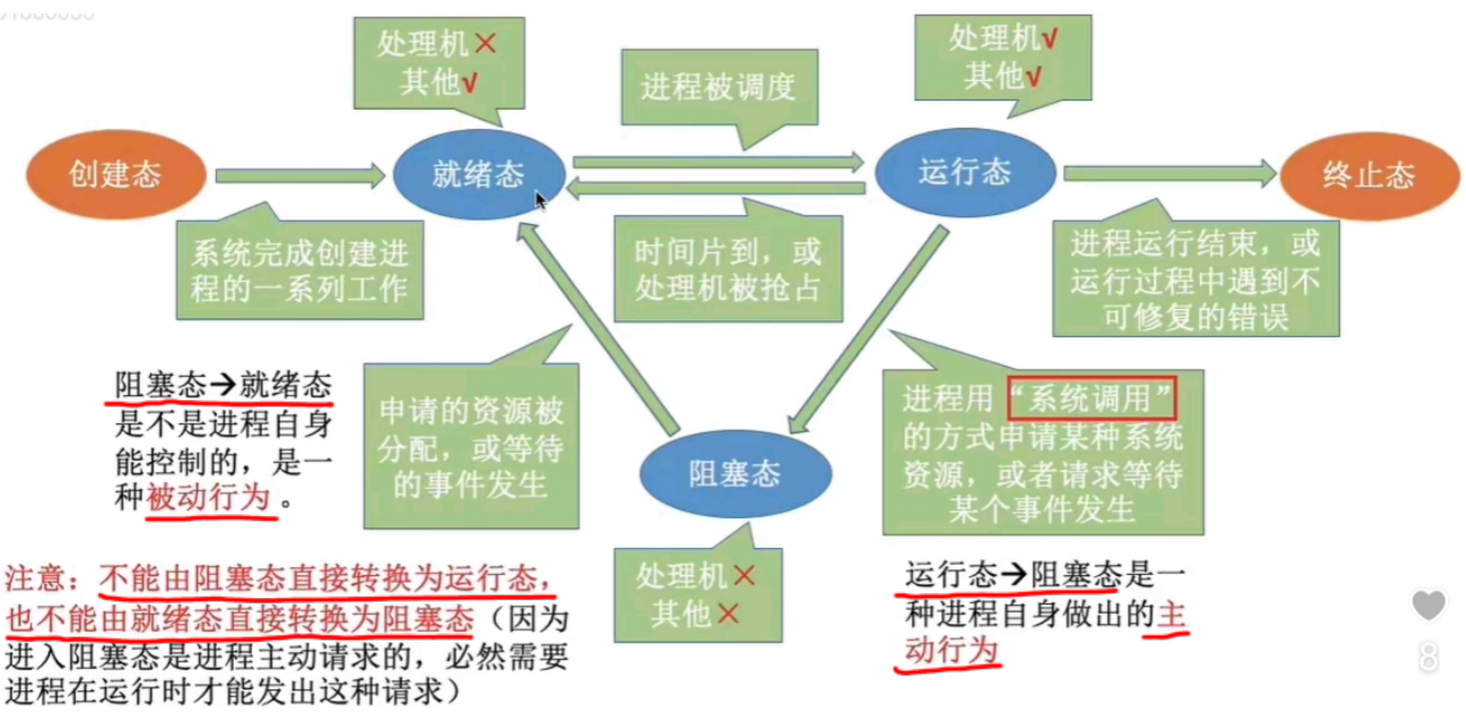
第二章 进程与线程

①进程

由三部分组成：程序段、数据段、进程控制块PCB

给进程　　给OS

进程的状态



只有运行→阻塞由进程自身决定

进程间通信

一个进程无法访问另一进程的地址空间

1）互斥地访问共享存储空间（基于数据结构/存储区）

2）消息传递（直接/间接）

直接通信方式是将消息直接挂到接受进程的消息队列里。

间接通信方式是将消息先发送到中间体。

3）管道通信（双向需2个管道）

一个管道只能有**一个读进程**，但能有**多个写进程**。当管道为空时，读进程阻塞；当管道不为空时，写进程阻塞。

②线程

各线程间可以并发，一个进程内的多个线程也可以并发

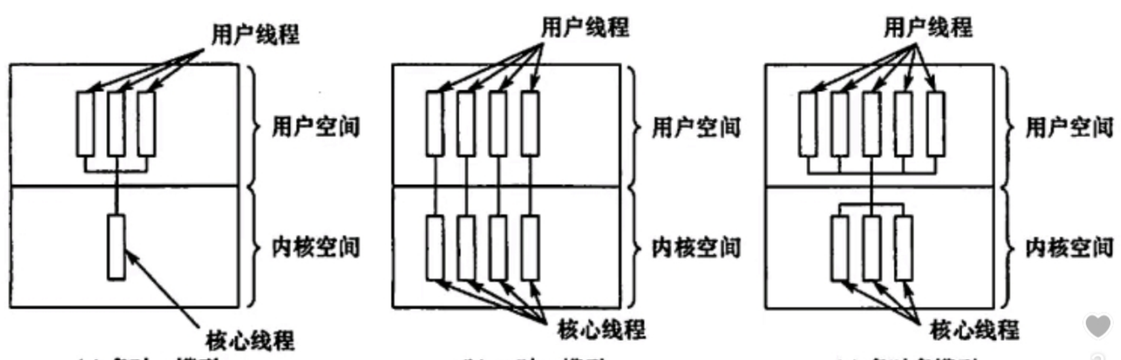
	进程	线程
资源	资源分配的基本单位哦	资源调度的基本单位
并发性	只能进程间并发	进程内也可以并发
系统开销	开销大	开销小
地址空间和资源	每个进程之间独立的地址空间和资源	共享地址空间和资源

管理　CPU状态

用户级线程　应用程序　用户态　线程阻塞导致整个进程阻塞
内核级线程　OS内核　核心态　一线程阻塞，其他线程仍执行
(通过线程控制块TCB)

多线程模型

	一对一	多对一	多对多
内容	将每个用户级线程映射到一个内核级线程	多个用户级线程映射到一个内核级线程	多个用户级线程映射到多个内核级线程上
优点	当一个线程被阻塞后，允许调度另一个线程运行，并发能力强	线程管理是在用户空间上进行的，效率比较高	克服多对一的缺点，有客服一对一开销大的缺点
缺点	每创建一个用户级线程，相应就需要创建一个内核级线程，开销大	如果一个线程阻塞，则都会发送阻塞，只允许一个线程运行	/



③处理机调度

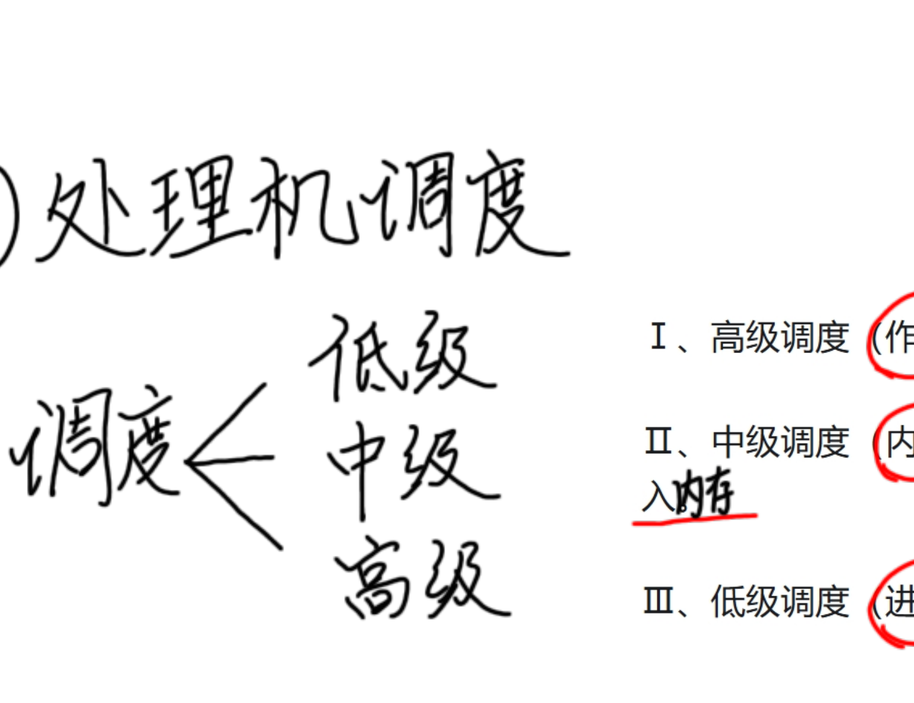
调度 ← 低级 中级 高级

Ⅰ、高级调度（作业调度）：从后备队列中调入一个作业进入就绪队列中。

Ⅱ、中级调度（内存调度）：中级调度实际上是外存与内存之间的调度。把进程从外存调入内存

Ⅲ、低级调度（进程调度）：从就绪队列中选取一个进程，然后使其由就绪态变为执行态。

调度算法指标



周转时间反映等待+运行时间

带权周转时间 ≥1，越低越好

先来先服务FCFS

用于作业调度 → 后备队列

用于进程调度 → 就绪队列

非抢占式、不会导致饥饿

短作业优先 SJF

非抢占式、可能饥饿/饿死

最短剩余时间优先SRTN是抢占式

高响应比优先

$$响应比 = \frac{等待时间 + 要求服务时间}{要求服务时间}$$

非抢占式、不会饥饿

等待时间相同时，要求服务时间短的优先(SJF 的优点)
要求服务时间相同时，等待时间长的优先(FCFS 的优点)

优先级调度

动态优先级/静态优先级

一般来说：系统进程>用户进程；交互型>非交互型；I/O型进程>计算型进程。

可能导致饥饿

非抢占式的优先级调度算法:每次调度时选择当前已到达且优先级最高的进程。当前进程主动放弃处理机时发生调度。

抢占式的优先级调度算法:每次调度时选择当前已到达且优先级最高的进程。当前进程主动放弃处理机时发生调度。另外，当就绪队列发生改变时也需要检查是会发生抢占。

时间片轮转调度

只用于进程调度(只有作业放入内存并分配进程后，才能分配时间片)

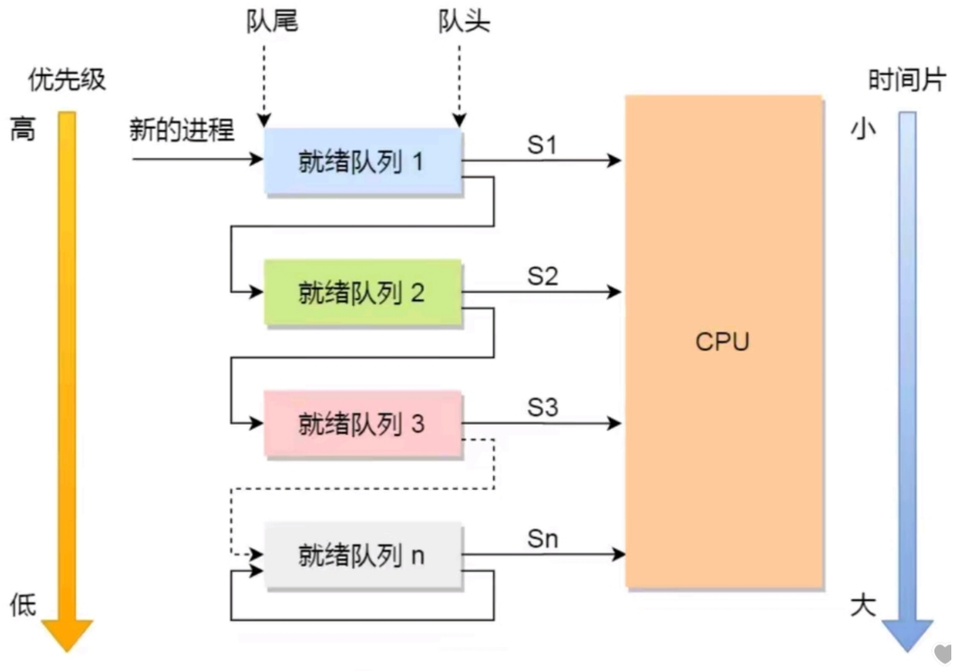
抢占式、不会饥饿

（常考）时间片太长或太短会有什么影响?

如果时间片**太大**，使得每个进程都可以在一个时间片内就完成，则**时间片轮转调度算法退化为先来先服务调度算法**，**并且会增大进程响应时间**。

另一方面，进程调度、切换是有时间代价的(保存、恢复运行环境)，因此如果时间片**太小**，会导致进程切换过于频繁，**系统会花大量的时间来处理进程切换**，从而导致实际用于进程执行的时间比例减少。可见时间片也不能太小。

多级反馈队列调度



（时间片：S1 < S2 < S3）

综合了多个算法的优点

- FCFS算法的优点是公平
- SJF算法的优点是能尽快处理完短作业，平均等待/周转时间等参数很优秀
- 时间片轮转调度算法可以让各个进程得到及时的响应
- 优先级调度算法可以灵活地调整各种进程被服务的机会

④ 互斥与同步

互斥：直接制约关系

同步：间接制约关系

两进程同时进入临界区

空闲让进。临界区空闲时，可以允许一个请求进入临界区的进程立即进入临界区。
忙则等待。当已有进程进入临界区时，其他试图进入临界区的进程必须等待。
有限等待。对请求访问的进程，应保证能在有限时间内进入临界区。
让权等待。当进程不能进入临界区时，应立即释放处理器，防止进程忙等待。

临界互斥的方法

软件实现：皮特森算法

```
bool flag[2];  //表意愿
int turn;      //表谦让
```

对P0进程

```
flag[0] = true;
turn = 0;
while(flag[1] && turn ==0); //P1有意愿且不想谦让，
                            P0想谦让，则等待
……
flag[0] = false;
```

对P1进程

```
flag[1] = true;
turn = 1;
while(flag[0] && turn ==1); // P0有意愿不谦让，P1谦让，则等待
……
flag[1] = false;
```

单标志法：违背空闲让进

算法思想：两个进程在访问完临界区后会把使用临界区的权限转交给另一个进程。也就是说每个进程进入临界区的权限只能被另一个进程赋予。turn的背后逻辑是表示谦让。

双标志法先检查：违背忙则等待（检查对方/切换自己flag有时间差）

算法思想：设置一个布尔型数组flag[]，数组中各个元素用来标记各进程想进入临界区的意愿，比如“flag[0]=ture”意味着0号进程P0现在想要进入临界区。每个进程在进入临界区之前先检查当前有没有别的进程想进入临界区，如果没有，则把自身对应的标志flag[i]设为tue,之后开始访问临界区。flag背后逻辑是表示意愿。

双标志法后检查：违反空闲让进、忙则等待可能会饥饿

算法思想：双标志先检查法的改版。前·个算法的问题是先“检查”后“上锁”，但是这两个操作又无法气阿成，因此导致了两个进程同时进入临界区的问题。因此，人们又想到先“上锁”后“检查”的方法，来避免上述问题。

硬件实现：中断屏蔽：只对当前CPU有效，不适于多处理机

硬件指令：不满足让权等待，可能会饥饿

⑤信号量

信号量只能被原语 wait(S)，signal(S)访问

↓ P(获取)　↓ V(释放)

记录型信号量

```
typedef struct{
    int value;
    struct process *L;  //该类资源的等待队列
} semaphore;
```

P操作

```
void wait(semaphore S) //相当于申请资源
        S.value --;
        if (S.value <0){
              add this process to S.L;
              block(S.L);
        }
}
```

//当资源分配完毕时，将自我阻塞

V操作

```
void signal(semaphore S) //相当于释放资源
         S.value ++;
         if (S.value <=0){
               remove a process P from  S.L;
               wakeup(P);
         }
}
```

//当释放后仍S.value <=0，则 S.L仍有被阻塞进程，因此需要唤醒S.L第一个进程使其使用释放的资源

实现同步

do P0;　V(S);
(前操作)

P(S);　do P1;
(后操作)

实现互斥

S=1;

P(S);　do P0;　V(S);

P(S);　do P1;　V(S);

实现互斥的P操作必须在同步的P操作之后，否则会死锁

生产者—消费者问题

初始化 semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0;

生产者 produce product;
P(empty); //获取空缓冲区
P(mutex); //进临界区
add product to buffer;
V(mutex);
V(full); //满缓冲区增多

(先同步P，再互斥P)

消费者 P(full);
P(mutex);
remove product from buffer;
V(mutex);
V(empty);
consume product;

读者—写者问题

初始化 int count = 0; //记录有几个读者同时读
semaphore mutex = 1; //修改count时互斥
semaphore rw = 1; //读写互斥
semaphore w = 1; //写优先

读者 P(w); //没有写进程时，才可继续
P(mutex); //互斥访问count
if(count==0) P(rw) //首进程读时，阻止写进程
count++;
V(mutex);
V(w);
reading;
P(mutex);
count--;
if(count==0) V(rw); //无读进程时，允许写进程
V(mutex);

写者： P(w);
P(rw);
writing;
V(rw);
V(w);

哲学家进餐问题

初始化：semaphore chop[5] = {1,1,1,1,1};
semaphore mutex = 1;

哲学家： P(mutex); //互斥取筷子
P(chop[i]); //先取左边
P(chop[(i+1)%5]); //再取右边
V(mutex);
eat;
V(chop[(i+1)%5]);
V(chop[i]);

⑥死锁

产生条件

①互斥条件：资源是互斥使用的

②不剥夺条件：进程只能由自己释放

③请求和保持：已获得资源同时，还在请求另外一个资源

④循环等待：存在一个循环等待链

预防

	内容	缺点
破坏互斥条件	将临界资源改造为可共享使用的资源（如SPOOLing技术）	可行性不高，很多时候无法破坏互斥条件
破坏不剥夺条件	方案一，申请的资源得不到满足时，立即释放拥有的所有资源 方案二，申请的资源被其他进程占用时，由操作系统协助剥夺（考虑优先级）	缺点：实现复杂；剥夺资源可能导致部分工作失效； 反复申请和释放导致系统开销大；可能导致饥饿
破坏请求和保持条件	运行前分配好所有需要的资源，之后一直保持	资源利用率低；可能导致饥饿
破坏循环等待条件	给资源编号，必须按编号从小到大的顺序申请资源（申请资源的顺序）	不方便增加新设备；会导致资源浪费；用户编程麻烦

银行家算法：

安全序列$\{P_1, P_2, \cdots P_n\}$

安全状态一定非死锁，不安全状态仅是可能死锁

算法尝试预分配，再执行安全检测，直至所有进程均在安全序列

安全性算法：

进程	最大需求	已经分配	最多需要
P0	(7,5,3)	(0,1,0)	(7,4,3)
P1	(3,2,2)	(2,0,0)	(1,2,2)
P2	(9,0,2)	(3,0,2)	(6,0,0)
P3	(2,2,2)	(2,1,1)	(0,1,1)
P4	(4,3,3)	(0,0,2)	(4,3,1)

此时剩余(3,3,2)

能满足P1，P3，则运行P1，剩余(5,3,2)，运行P3，剩余(7,4,3)

进程	最大需求	已经分配	最多需要
P0	(7,5,3)	(0,1,0)	(7,4,3)
P2	(9,0,2)	(3,0,2)	(6,0,0)
P4	(4,3,3)	(0,0,2)	(4,3,1)

P0，P2，P4均可满足，因此安全

检测：资源有向图是否成环

解除

①资源剥夺：从死锁进程处抢夺资源。

②撤销进程法：强制撤销部分或全部死锁，并剥夺这些进程的资源。

③进程回退法：让进程回退到足以避免死锁的时候。

第三章 内存管理

① 相关概念

$1K = 2^{10}$ ，$1M = 2^{20}$，$1G = 2^{30}$

按字节编制，每个存储单元1字节，8bit

按字编制，每个存储单元1字，64 bit

② 内存分配

1）连续分配

内部碎片：已经分配　　外部碎片：还未分配

内部碎片是处于区域内部或页面内部的存储块。占有这些区域或页面的进程并不使用这个存储块。而在进程占有这块存储块时，系统无法利用它。直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

外部碎片是出于任何已分配区域或页面外部的空闲存储块。这些存储块的总和可以满足当前申请的长度要求，但是由于它们的地址不连续或其他原因，使得系统无法满足当前申请。

单一连续分配：内存中只可有一个用户程序

无外部碎片，有内部碎片

固定分区分配：每个小分区只有一个用户程序

无外部碎片，有内部碎片

动态分区分配：

算法	优点		优点	缺点
首次适应	综合看性能最好，算法开销小（不能排列地址）		综合性能最好。算法开销小，不需要对空闲分区进行排序	\
最佳适应	优先使用更小的分区	按照空闲分区容量递增的次序	会有更大的分区被保留下来，更能满足大进程的需求	会产生很对难以利用的碎片，算法开销大，因为要重新对空闲分区进行排序
最坏适应	优先使用更大的分区	按照空闲分区容量递减的次序	可以减少难以利用的碎片	大分区很容易被用完，不利于大进程，而且算法开销大
邻近适应	首次适应后，每次查找从上次位置继续查找	空闲分区以地址递增次序排列	不用每次都从低地址开始检索，算法开销小	会使高地址的大分区也被用完

2）非连续分配

分页存储管理

页表包括页号和块号

页号=逻辑地址/页面长度（取除法的整数部分）

页内偏移量=逻辑地址%页面长度（取除法的余数部分）

相联存储器(TLB)在Cache中，页表在内存中

分段存储管理

段表包含段号和主存起始地址

页是信息的**物理单位**。分页的主要日的是为了实现离散分配，提高内存利用率。分页仅仅是系统管理上的需要，完全是系统行为，对用户是不可见的。

段是信息的**逻辑单位**。分段的主要目的是更好地满足用户需求。一个段通常包含着一组属于一个逻辑模块的信息。分段对用户是可见的，用户编程时需要显式地给出段名。

页的大小固定且由**系统决定**。

段的长度却不固定，**取决于用户编写的程序。**

分页的用户进程地址空间是**一维**的

分段的用户进程地址空间是**二维**的

段页式管理

地址空间先分段，再分页

内存空间划分为页大小的块，以块为单位分配内存

逻辑地址(段号，页号，页内偏移量)

访存次数：查段表、查页表、访问目标存储单元

每个段对应一个段表项。各段表项长度相同，由段号（隐含）、页表长度、页表存放地址 组成

每个页对应一个页表项。各页表项长度相同，由页号（隐含）、页面存放的内存块号 组成

③虚拟内存管理

请求分页管理

相比分页管理，还会调入调出

页表

页号	内存块号	状态位	访问字段	修改位	外存地址
0	a	1	0	0	x
1	b	1	10	0	y
2	c	1	6	1	z

是否已调入内存　最近访问时间　是否被修改

若访问页面不在内存，产生缺页中断，置换新旧页面入内存

页面置换算法

	内容	优缺点
最佳置换算法（OPT）	优先淘汰最长时间不会被访问的	缺页率最好，性能最好，但无法实现
先进先出置换算法（FIFO）	优先淘汰最先进入内存的页面	实现简单，但性能很差，可能会出现Belady异常*
最近最久未使用置换算法（LRU）	优先淘汰最近最久没有被访问的页面	性能很好，但需要硬件支持，算法开销大，对所有页进行排序
时钟置换算法（CLOCK）	循环扫描各页面，第一轮淘汰访问位=0，并将扫描过的页面访问位改位0。若第一轮没选中，则进行第二轮扫描	实现简单，算法开销小；但未考虑页面是否被修改，修改后置换还要换出外存，这是主要的开销
改进型的时钟置换算法	第一轮:从当前位置开始扫描到第一个（0,0）的帧用于替换。本轮扫描不修改任何标志位 第二轮:若第一轮扫描失败，则重新扫描，查找第一个（0,1）的帧用于替换。本轮将所有扫描过的帧访问位设为0 第三轮:若第二轮扫描失败，则重新扫描，查找第一个（0,0）的帧用于替换。本轮扫描不修改任何标志位 第四轮:若第三轮扫描失败，则重新扫描，查找第一个(0,1）的帧用于替换	算法开销较小，性能也不错

页面分配策略

驻留集：给进程分配物理块的集合

太小，缺页频繁，出现抖动现象

太大，并发度下降

固定分配VS可变分配：区别在于进程运行期间驻留集大小是否可变。

局部置换VS全局置换：区别在于发生缺页时是否只能从进程自己的页面中选择一个换出。

可变分配局部置换：根据缺页率动态增减物理块

④内存相关题目

已知系统为32位实地址，采用48位虚拟地址，页面大小4KB，页表项大小为8B；每段最大为4GB。

（1）假设系统使用纯页式存储，则要采用多少级页表，页内偏移多少位？

（2）假设系统采用一级页表，TLB命中率为98%，TLB访问时间为10ns，内存访问时间为100ns，并假设当TLB访问失败后才访问内存，问平均页面访问时间是多少？

（3）如果是二级页表，页面平均访问时间是多少？

（4）上题中，如果要满足访问时间<=120ns，那么命中率需要至少多少？

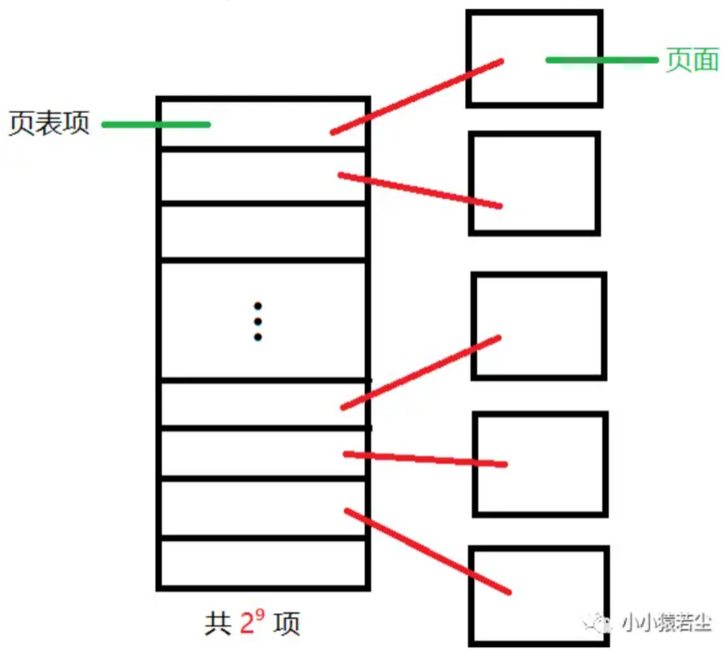
（5）若系统采用段页式存储，则每用户最多可以有多少段？段内采用几级页表？

(1) 页面大小4KB → 2^{12} bits → 页内偏移12位

虚拟地址48位 − 12位页内偏移 = 36位用于表示虚拟页号，2^{36}个页面

一个页面可放4KB/8B = 2^9个页表项，即一级页表可放2^9个页面

⇒ 36/9 = 4级（$2^9 \cdot 2^9 \cdot 2^9 \cdot 2^9$个页面）



一级页表

(2) 98%只需TLB中页表项+内存中页面

2%先TLB未命中，再从内存中找页表项，再从内存中访问页面

$98\% \times (10+100) + 2\% \times (10+100+100) = 112\,ns$

(3) 如果TLB命中，情况同(2)

若TLB未命中，需访问2次内存中页表+1次内存中页面

$98\% \times (10+100) + 2\% \times (10+100+100+100) = 114\,ns$

(4) TLB命中率t

$t \times (10+100) + (1-t) \times (10+100+100+100) \leqslant 120\,ns$

(5) 每段4GB → 2^{32} B → 段内偏移32位

段号位数 = 虚拟地址48位 − 段内偏移32位 = 16位

每个用户有2^{16}个段

4GB段内纯页式存储，共32位，32位−页内偏移12位 = 20位页号

每页可存放页表项 4KB(页大小)/8B(页表项大小) = 2^9个

因此$\lceil 20/9 \rceil$ = 3级页表

【2021 统考真题】某请求分页存储系统的页大小为 4KB，按字节编址。系统给进程 P 分配 2 个固定的页框，并采用改进型 Clock 置换算法，进程 P 页表的部分内容见下表。

页号	页框号	存在位 1：存在，0：不存在	访问位 1：访问，0：未访问	修改位 1：修改，0：未修改
…	…	…	…	…
2	20 H	0	0	0
3	60 H	1	1	0
4	80 H	1	1	1
…	…	…	…	…

若 P 访问虚拟地址为 02A01H 的存储单元，则经地址变换后得到的物理地址是（

页大小4KB → 12位页内偏移 → A01H偏移，02页号

页号2存在位为0，不在主存需移入，已放满故需置换

3，4页都被访问过，3未修改，所以移走3号页面，地址为60H

第四章 文件管理

① 文件分类

无结构文件

流式文件，如.txt文件

有结构文件

1）顺序文件

分为顺序/链式存储，链式和可变长的顺序存储无法随机存取

串结构无法找到某关键字对应记录，顺序结构可以

2）索引文件

可随机存取，方便增删，索引表很占空间

3）索引顺序文件

先查索引表找到分组，再组内顺序查找

题目：有一个顺序文件含有10000个记录，平均查找的记录数为5000个，采用索引顺序文件结构，则最好情况下平均只需查找()次记录。

最好情况分$\sqrt{n}$组，每组$\sqrt{n}$个记录，共计$\sqrt{n}\cdot\sqrt{n}=n$个记录

索引表内顺序查找平均$\frac{\sqrt{n}}{2}$次，组内顺序查找平均$\frac{\sqrt{n}}{2}$次

$\Rightarrow \frac{\sqrt{10000}}{2}+\frac{\sqrt{10000}}{2}=100$次

② 文件目录

一个文件对应一个文件控制块FCB，一个FCB即一个目录项，多个目录项构成目录文件

目录结构

I、单级目录：一个系统只有一张目录表，不允许文件重名。

II、两级目录：不同用户的文件可以重名，但不能对文件进行分类。

III、多级（树型）目录：不同目录下文件可以重名，可以对文件进行分类，从根目录出发是“绝对路径”，从当前目录出发的路径是“相对路径”。

IV、无环图目录路径：在树型目录的基础上，增加一些指向同一节点的有向边，便于共享，一个实际存在外存的文件，可以被多个文件目录指向。删除时，为共享文件设置一个计数器（用来表明有多少个文件被共享），计数器为0时才真正删除该结点。

③文件物理结构

1)连续分配

物理块号=起始块号+逻辑块号

优点：支持顺序访问和直接访问，连续分配文件在读/写时速度最快

缺点：①连续分配当文件扩展时，十分不方便。

②连续分配会导致存储空间利用率低，会产生难以利用的空间碎片。

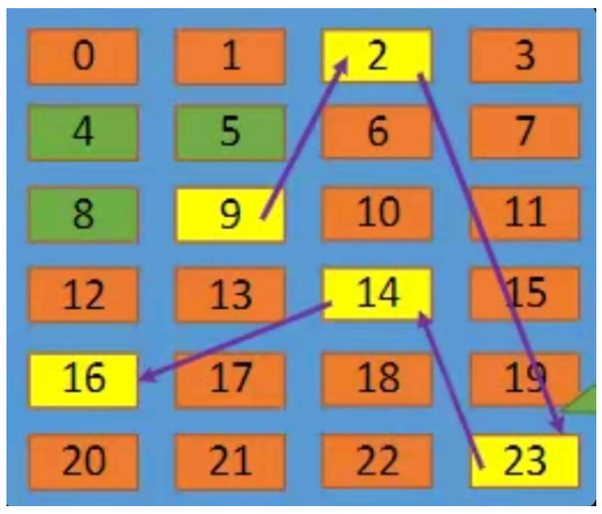
2)链接分配

隐式链接

逻辑块号i，在目录项中找到起始块号，再顺序查找，共计i次访问

优点：方便文件拓展，不会有碎片问题，外存利用率高。

缺点：只支持顺序访问，不支持随机访问，查找效率低，指向下一个盘块的指针也需要消耗少量的存储空间。

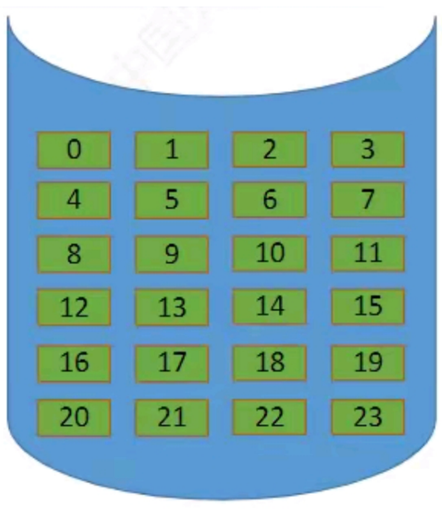


显式链接

FAT文件分配表常驻内存

优点：很方便文件拓展，不会有碎片问题，外存利用率高。并支持**随机访问**。相对于隐式来说，**地址转换不需要访问磁盘，因此文件的访问效率更高**。只需要访问在内存中的FAT。

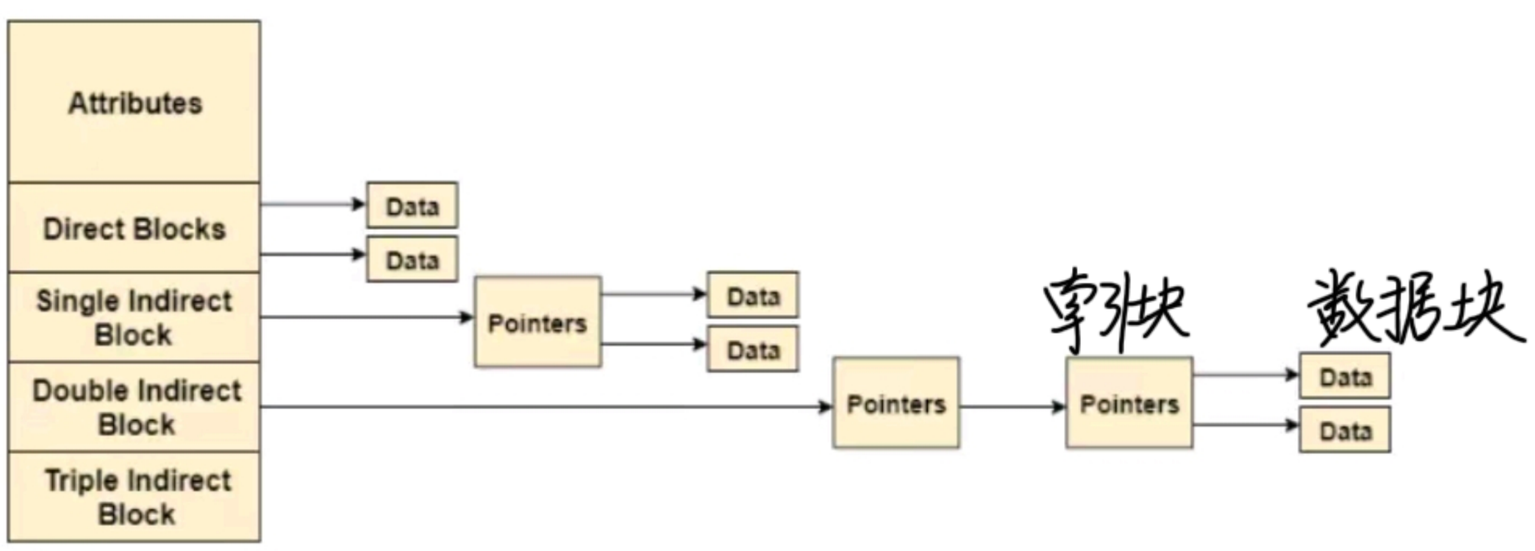
缺点：文件分配表的需要占用一定的存储空间。



物理块号	下一块
0	1
1	-1
2	5
3	-1
4	23
5	0
.....	
22	
23	3

FAT（文件分配表）

3)索引分配



计算文件最大长度

磁盘块大小为1KB，索引表项4B，则一级索引 $\frac{1KB}{4B}=256$ 个数据块

⇒ 2级索引时，256×256×1KB

地址转换

访问1026号逻辑块，一级索引的1026/256=4号表项，再二级索引的1026%256=2号表项

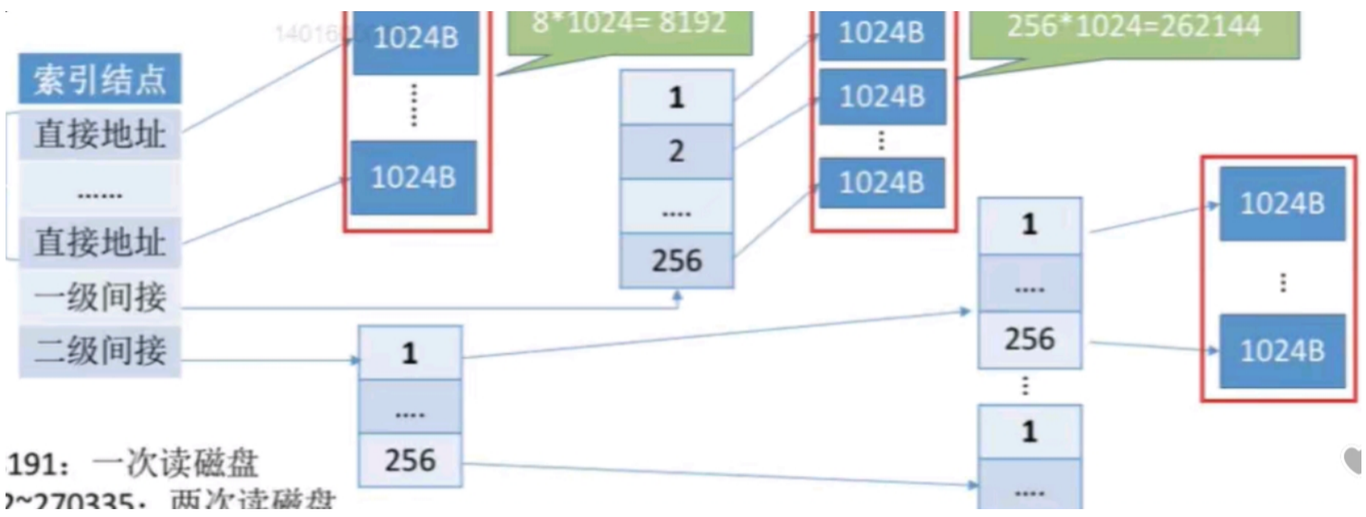
1.在文件的索引节点中存放直接索引指针10个，一级和二级索引指针各1个。磁盘块大小为1KB,每个索引指针占4个字节。**若某文件的索引节点已在内存中**，则把该文件偏移量（按字节编址)为1234和307400处所在的磁盘块读入内存，需问的磁盘块个数分别是？

无需从磁盘调入内存

8个直接索引，即 0～8×1KB 仅需一次外存访问

1个一级索引，指向了1KB/4B=256个数据块，即
8×1KB～8×1KB+256×1KB 仅需两次访问

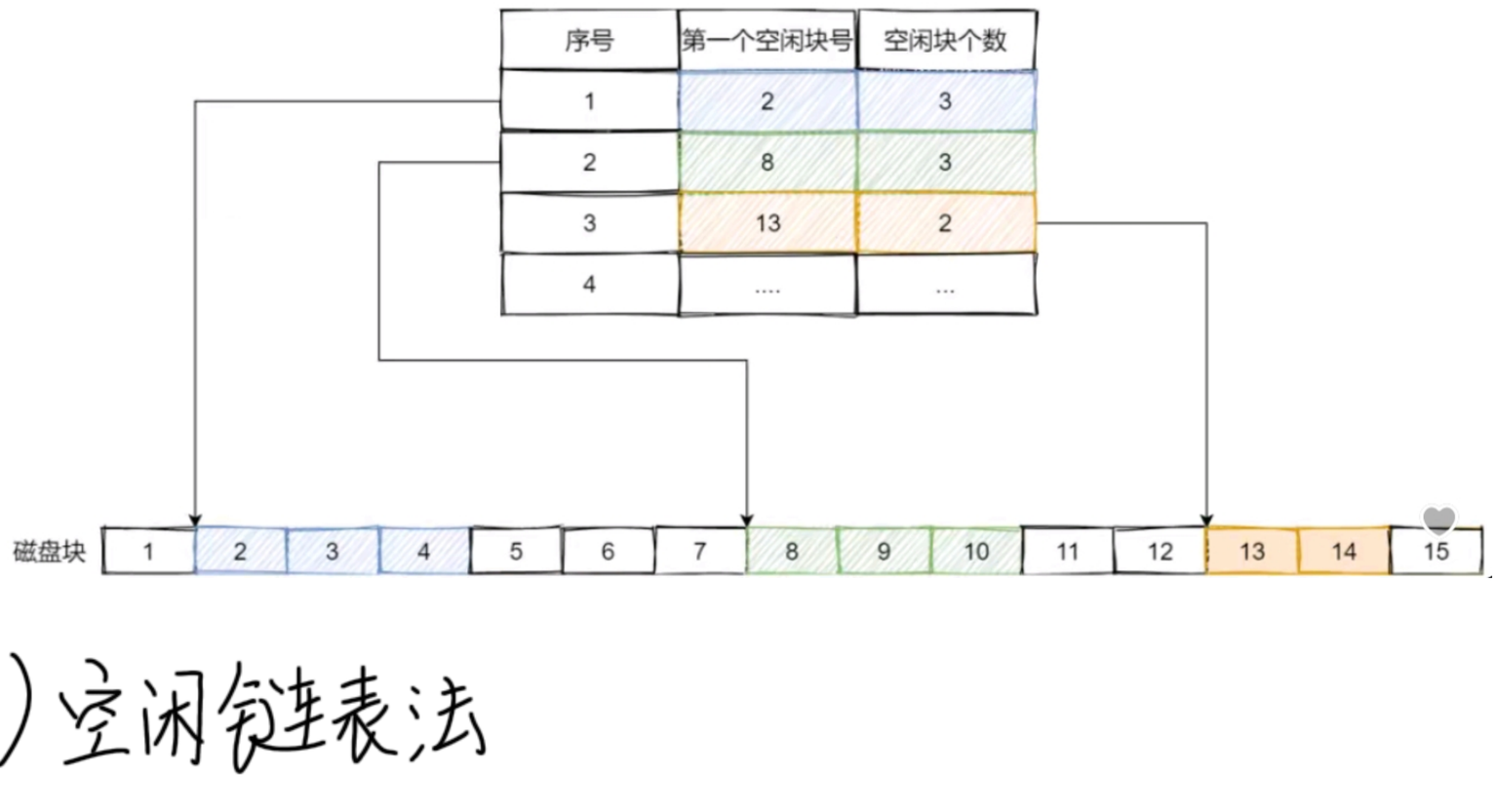
1个二级索引，即 8×1KB+256×1KB～8×1KB+256×1KB+256×256×1KB



④文件存储空间

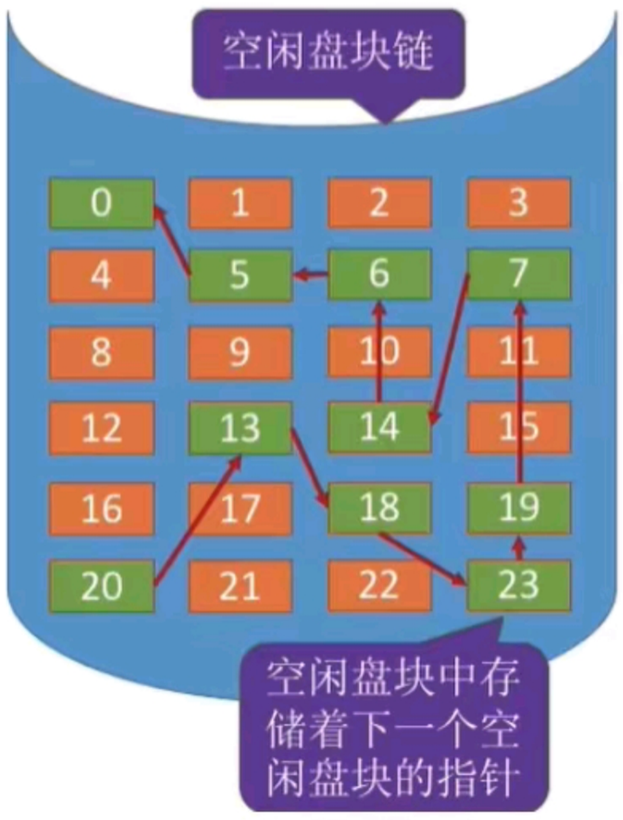
1)空闲表法

分配方式：首次/最佳/最坏适应等

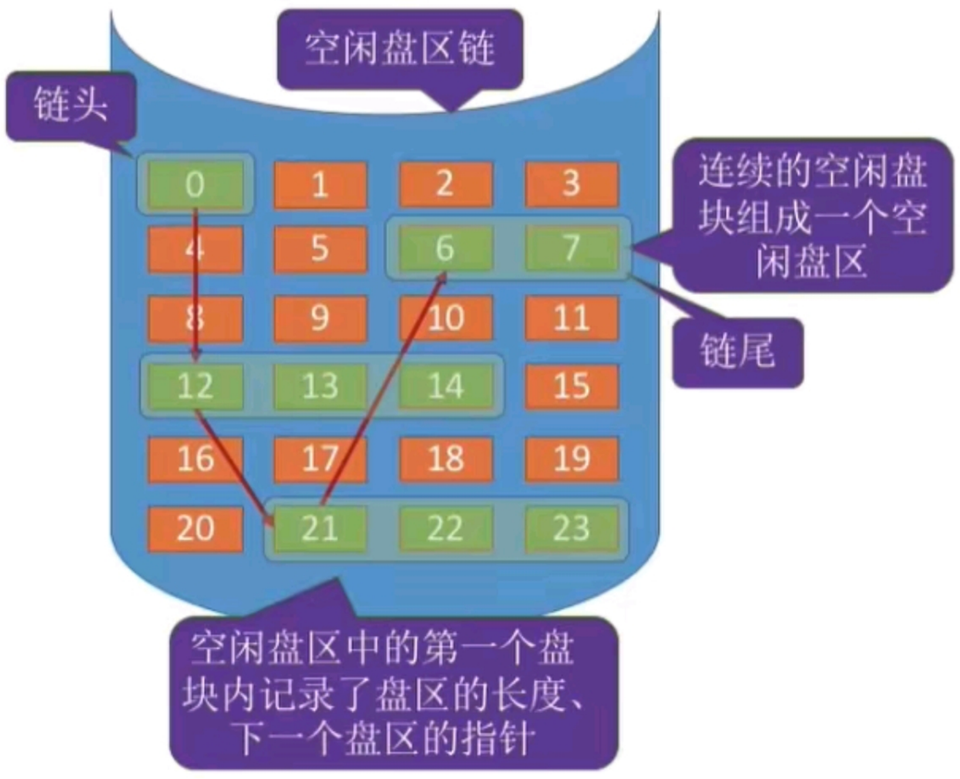


2)空闲链表法

空闲盘块链

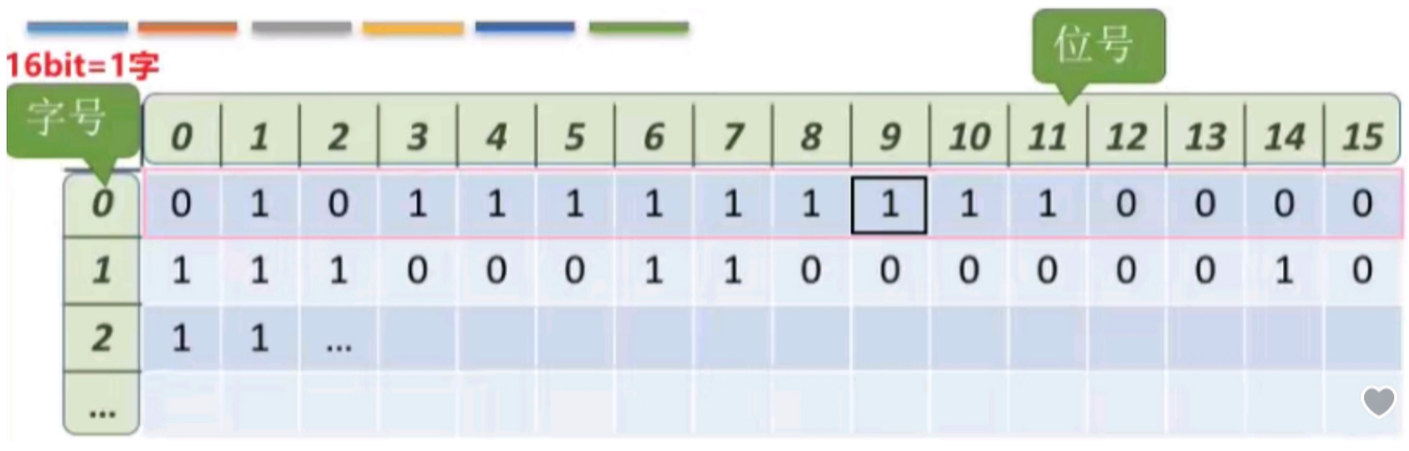


空闲盘区链



3) 位示图法

每个盘块均有一个二进制位与之对应



共n列

盘块号 = 字号 × 列数n + 位号

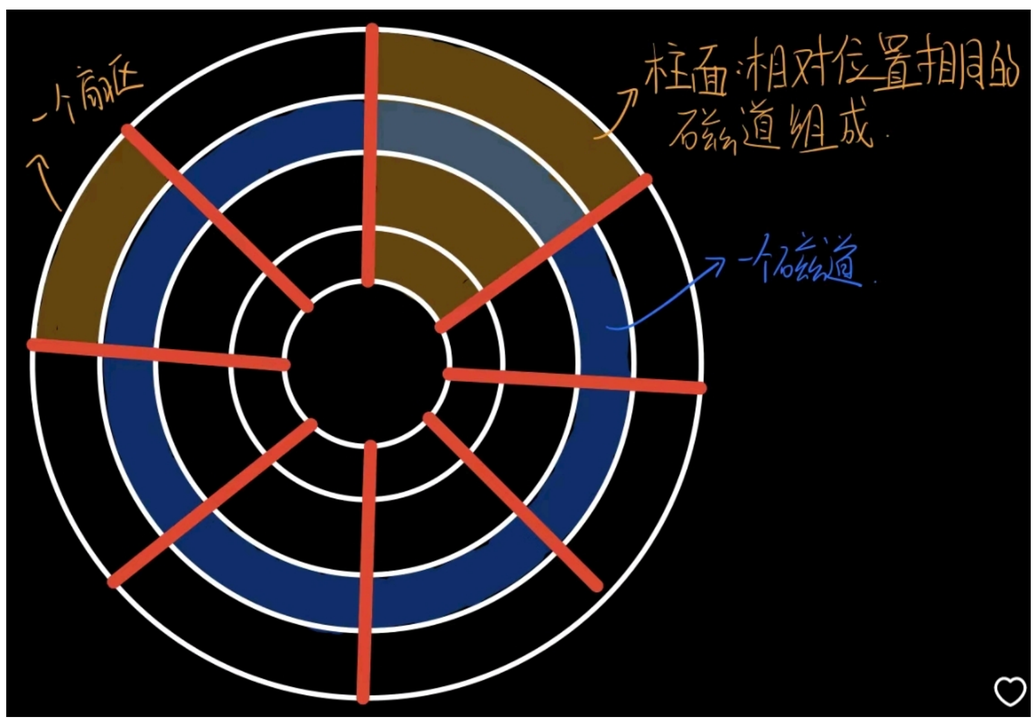
字号 = 盘块号 / n ; 位号 = 盘块号 % n ;

⑤ 文件保护

	内容	优缺点
口令保护	为文件设置一个口令，用户想要访问就必须提供口令	实现开销小，但是口令放在FCB不安全
加密保护	用一个密码对内容进行加密，访问时必须提供密码	安全性高，但加密/解密需要耗费一定的时间
访问控制	用一个ACL（访问控制表）记录各个客户对文件的访问权限	实现灵活，可以实现复杂文件的保护功能

⑥ 磁盘

磁盘结构

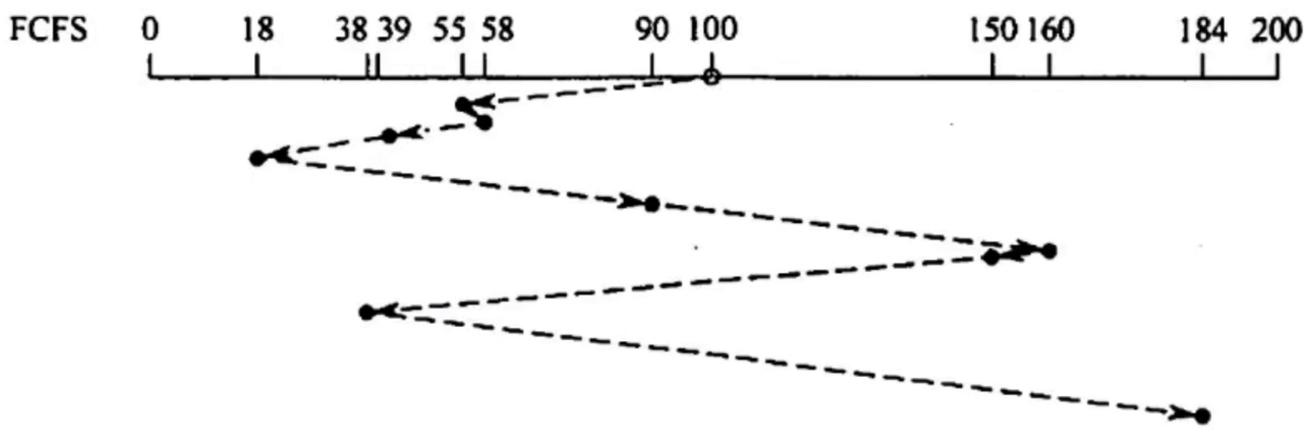


(柱面号，盘面号，扇区号)

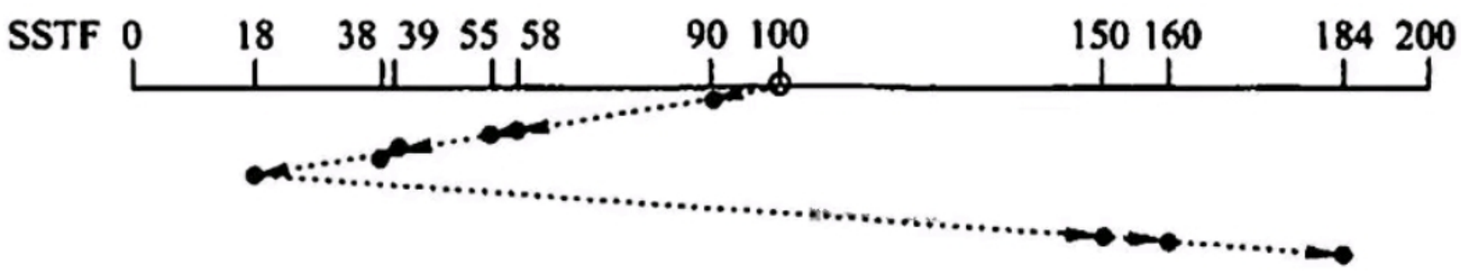
调度算法

访问时间 = 寻道 + 延迟 + 传输时间

1) 先来先服务 FCFS

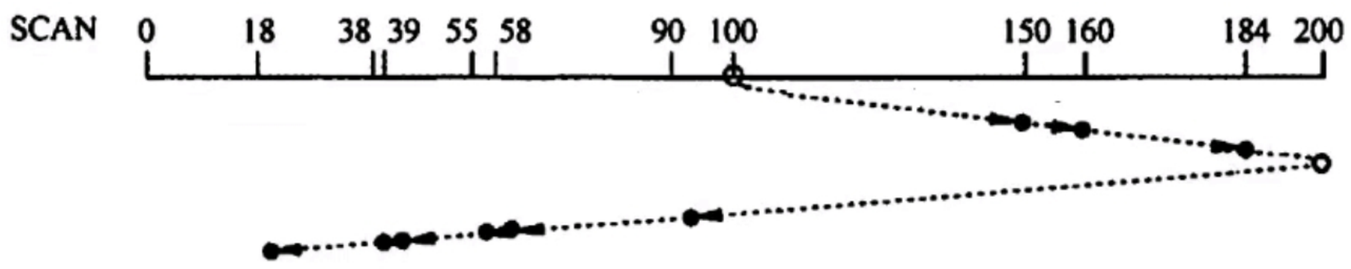


2) 最短寻找时间优先 SSTF (贪心)

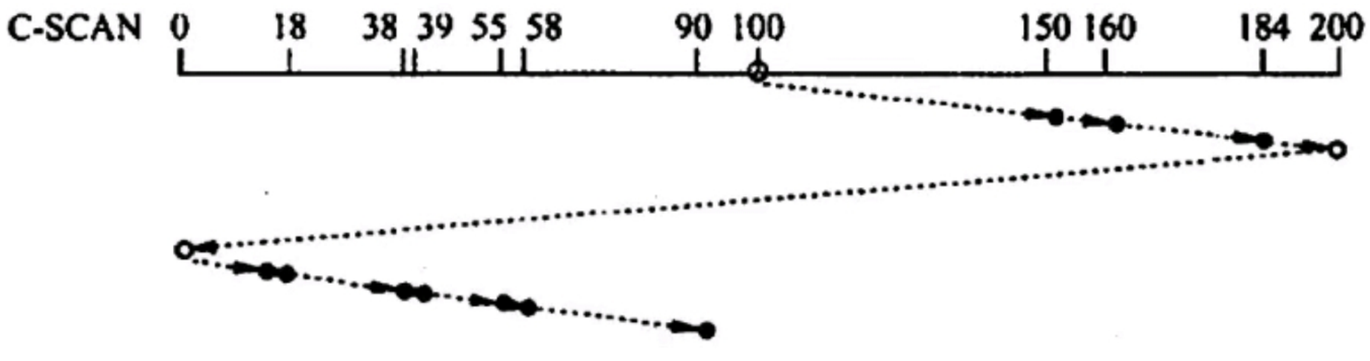


可能饥饿

3) 电梯/扫描算法 SCAN



4) 循环扫描 C-SCAN



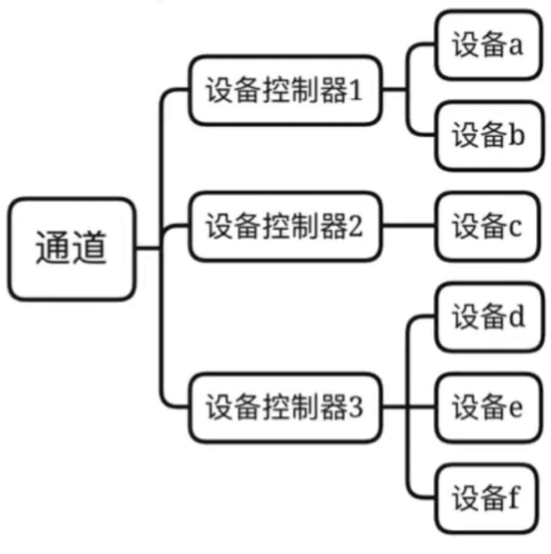
第五章 输入输出管理

①IO管理

I/O控制器：将CPU从繁杂IO事务中解脱

I/O控制方式：

	过程	CPU干预频率	传输单位	数据流向
程序直接控制方式	CPU不断轮询查看设备状态	极高	字	设备-CPU-内存 内存-CPU-设备
中断驱动方式	CPU指令执行完，查看有没有中断请求	高	字	设备-CPU-内存 内存-CPU-设备
DMA	CPU发出请求后就可以做其他事，剩下事由DMA控制器完成，完成后向CPU发送完成信号	中	块	设备-内存 内存-设备
通道控制方式	CPU发出请求后就可以做其他事，完成后向CPU发送完成信号	低	一组块	设备-内存 内存-设备



②设备独立性软件

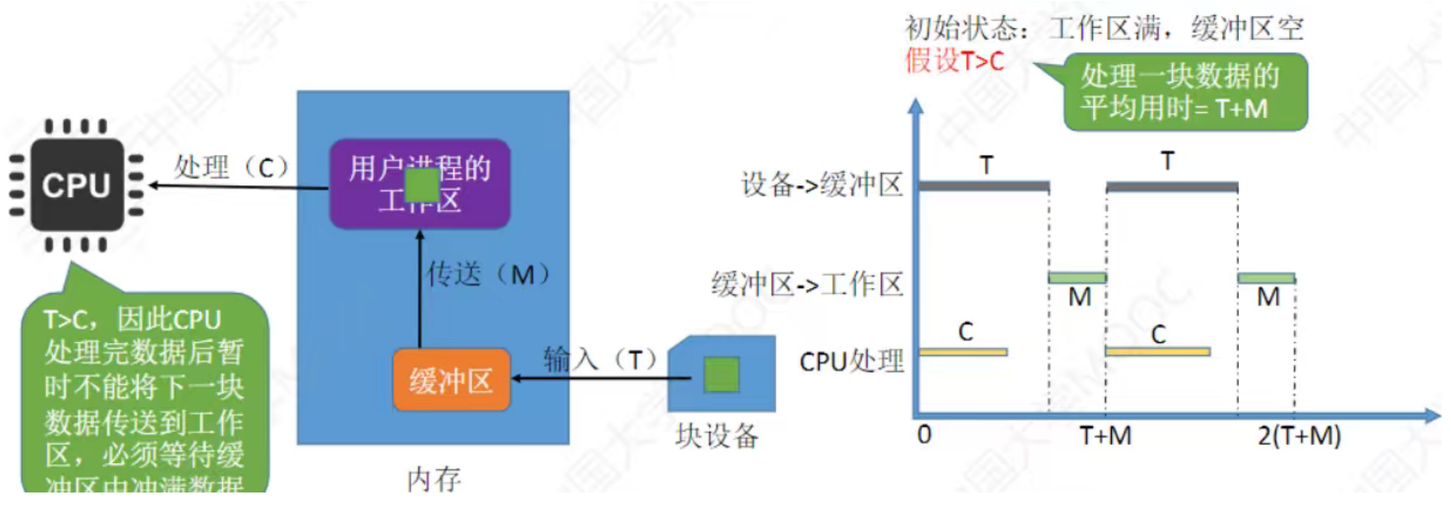
Ⅰ、目的：缓解CPU与设备的速度矛盾、减少对CPU的中断频率、解决数据粒度不匹配的问题、提高CPU与/IO设备之间的并行性。

1)单缓冲

一个缓冲区就是一个数据块

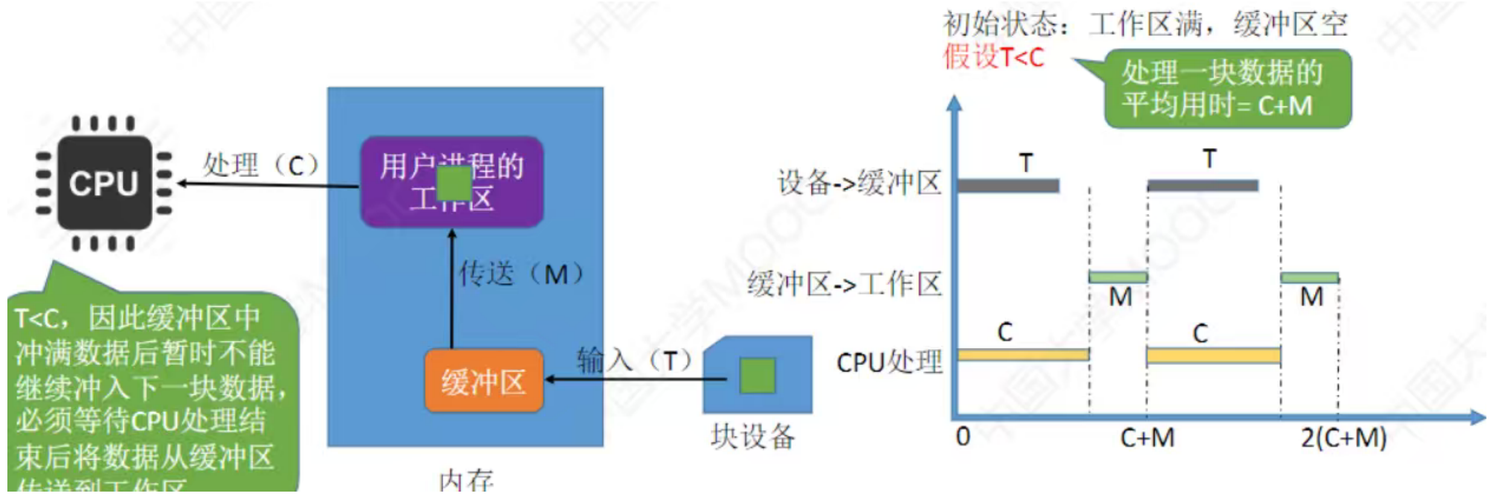
必须完满后，才能从缓冲区传出

处理一块数据平均耗时：当输入时间T>处理时间C，



平均耗时 T+M

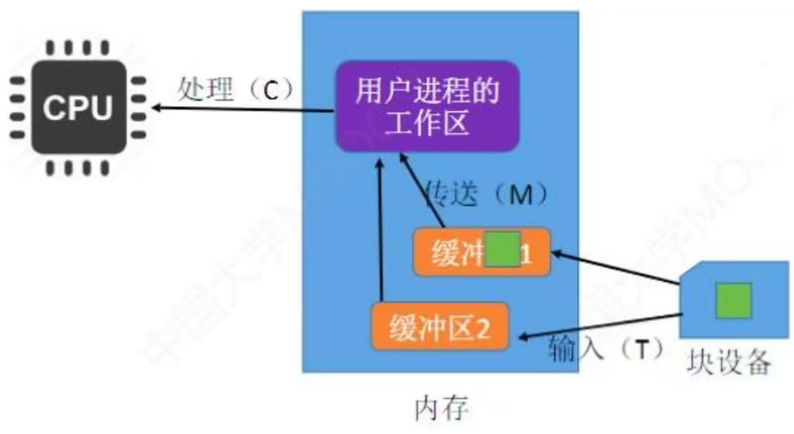
当输入时间T<处理时间C



平均耗时 C+M

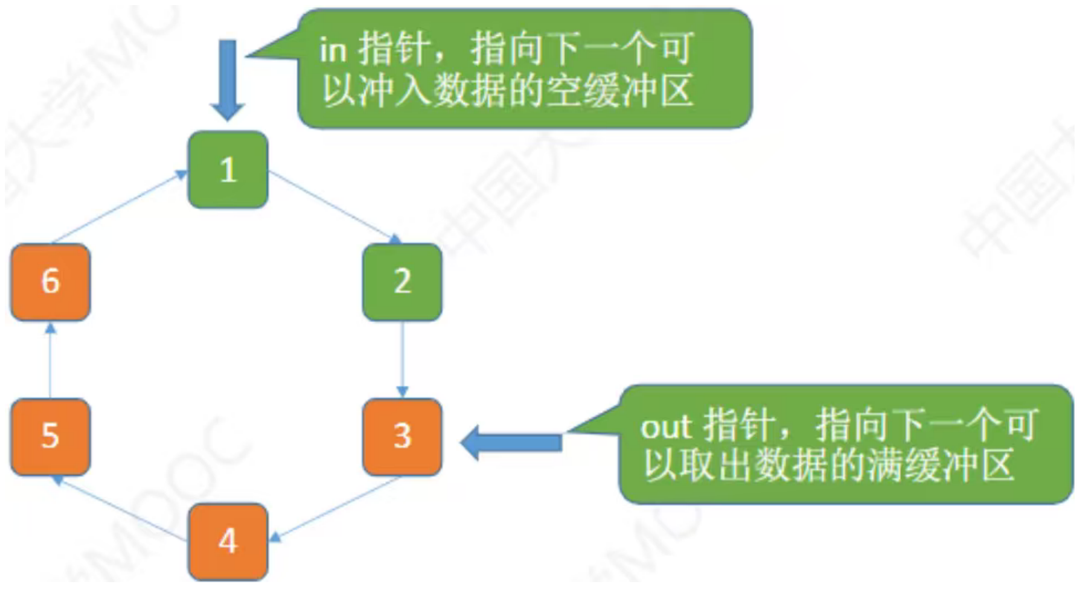
⟹单缓冲时，处理一块数据平均耗时 Max(C,T)+M

2) 双缓冲



⟹处理一块数据平均耗时为 Max(C+M,T)

3) 循环缓冲



4) 缓冲池

缓冲区 vs Cache

		高速缓存	缓冲区
相同点		都介于高速设备和低速设备之间	
区别	存放数据	存放的是低速设备上的某些数据的复制数据，即高速缓存上有的，低速设备上面必然有	存放的是低速设备传递给高速设备的数据（或相反），而这些数据在低速设备（或高速设备）上却不一定有备份，这些数据再从缓冲区传送到高速设备（或低速设备）
	目的	高速缓存存放的是高速设备经常要访问的数据，若高速设备要访问的数据不在高速缓存中，则高速设备就需要访问低速设备	高速设备和低速设备的通信都要经过缓冲区，高速设备永远不会直接去访问低速设备

SPOOLing假脱机技术

对打印机，先输出到磁盘的输出井中，再传向打印机