1. **绪论**

**§1 OS\_Introduction/OS\_History/CS\_Introduction**

**·操作系统作用**

1. 用户与计算机硬件系统之间的接口
2. 计算机资源的管理者
3. 扩充机器

**·操作系统定义**

操作系统是一组控制和管理计算机软硬件资源、合理地对各类作业进行调度以及方便用户的程序的集合。

**·经典的操作系统体系结构——层次结构**

**最高层：接口**

**中间层：对对象进行操纵和管理的软件集合**

**最底层：OS操纵和管理的对象，包括各类软硬件**

**·操作系统设计目标**

1. 方便性 使执行程序、解决问题更简单，让计算机系统更方便使用
2. 有效性 提高软硬件资源利用率
3. 可扩充性 适应软硬件发展需求
4. 开放性 可移植性、互操作性

**操作系统最重要两个目标：方便性、有效性**

**·操作系统的历史（10年→10年→八零年）**

1945~1955 无操作系统

1955~1965 批处理系统

1965~1980 引入分时

1980~present PC时代，百花齐放

1990~present 移动计算时代

**·罗列几个操作系统（随便记几个）**

1. FMS
2. OS/360
3. UNIX类、Linux
4. Windows
5. VxWorks

**·推动操作系统发展的主要动力**

1. 不断提高计算机资源利用率的需要

2. 方便用户

3. 器件的不断更新换代

4. 计算机体系结构的不断发展

**·批处理系统**

**批(batch)的含义**

供一次加载的磁带或磁盘，通常由若干个作业组装而成，在处理中使用一组相同的系统软件。

**单道批处理系统**

批处理系统的引入**目的：**提高系统资源的利用率和吞吐量

**特征：**自动性、顺序性、单道性

**处理过程：**监督程序（monitor）

系统对作业的处理都是成批进行的，且内存中始终只保持一道作业。

**当时主要矛盾：**CPU速度与I/O速度之间的矛盾

**解决办法：**Off-line I/O（脱机I/O）、磁盘（Disk）

**脱机I/O**

**目的：**解决人机矛盾和CPU与I/O设备之间速度不匹配的矛盾

**方法：**利用低速的外围机进行，纸带（卡片）*→*磁带（磁盘）

**人机矛盾：**人工操作方式与机器利用率的矛盾

**多道批处理系统**

**多道：**系统中同时驻留多个作业，共享内存，复用CPU（当一个作业因某个原因暂停运行时，切换到另一个作业上运行）

**优点**

1. 提高CPU利用率
2. 提高内存和I/O设备利用率
3. 提高系统吞吐量

**特征**

1. 多道性
2. 无序性
3. 调度性（作业调度、进程调度）

**·2阶段调度（2-phrase of scheduling）**

作业调度：决定将哪几个作业调入内存，并为作业创建进程、分配资源

进程调度：从内存的作业池中选取一个（或几个）进程分配 CPU 进行处理

**·分时系统**

**关键：**及时接收、及时处理

**特征：**多路性、独立性、及时性、交互性

**·实时系统**

**分类：**硬实时系统、软实时系统

**典型：**VxWorks、QNX、RTEMS

**·三种最基本操作系统类型**

1. 批处理系统
2. 分时系统
3. 实时系统

一个实际的操作系统往往兼有上述三种操作系统类型的功能

**·MBR：主引导记录（**Master Boot Record**）**

硬盘驱动器上的第一个扇区，一种特殊类型的引导扇区

**MBR** **=** **MBR code** (also called boot loader) **+** **partition table**

**·现代操作系统是中断驱动的（interrupt-driven）**

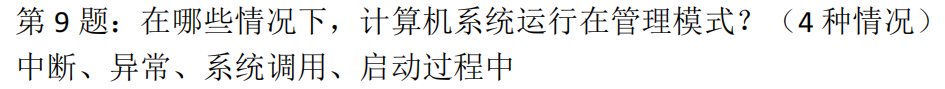
**·I/O方式：**同步、异步

**·存储层次之间的矛盾在于：**成本、速度、容量

**·Hardware protection**

1. Dual-Mode Operation （双模式）

提供两种模式：用户模式和特权模式（管理模式/系统模式/监督程序模式）

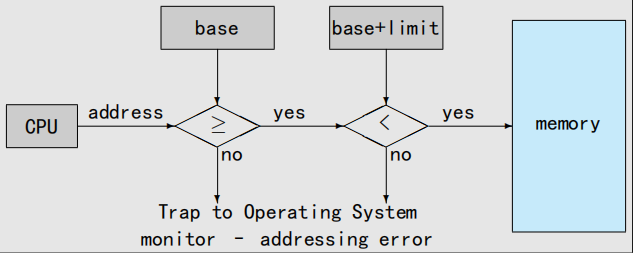


1. I/O protection

防止用户发出非法IO指令

所有IO指令都是特权指令，用户程序需要通过系统调用来进行IO操作

1. Memory protection



使用基址寄存器与限制寄存器，范围外的地址受保护无法访问

1. CPU protection

用户程序死循环？

使用定时器（timer），计时结束后产生中断，将控制权交给操作系统。

**§2 OS结构**

**·操作系统的组件/模块/功能**

1. Process Management, 进程管理
2. Main Memory Management, 内存管理
3. I/O System Management, I/O管理
4. File Management, 文件管理
5. Secondary-Storage Management, 辅存/外存管理
6. Command-Interpreter System, 命令解释系统
7. Protection System, 保护
8. Networking, 网络

**操作系统有哪些服务/系统调用？**回答同上，没区别

**·用户接口有哪几种？**

命令接口、程序接口、图形接口

**·Device independence(设备独立性)：**即程序与具体的设备无关。

**·操作系统提供服务的最基本方式：**system calls(系统调用)

**什么是系统调用号？**

每个系统调用有唯一的数字（系统调用号）与之对应，用以找到系统调用的入口。

**系统调用传递参数的三种方式？**

1. 参数存放在寄存器中，通过寄存器传递
2. 参数存放在内存中的块或表内，块或表的地址存放在寄存器中作为参数传递
3. 程序将参数入栈，再由OS将之出栈，完成参数传递

**·操作系统特征**

1. 并发

并行：两或多个事件在同一时刻发生。

并发：两或多个事件在同一时间间隔内发生。

1. 共享

系统中资源可供内存中多个并发执行的进程共同使用。

1. 虚拟

通过某种技术把一个物理实体变为若干个逻辑上的对应物。

1. 异步

运行进度不可预知。

**并发和共享是操作系统的两个最基本的特征。**

**·操作系统的结构**

1. Simple structure(无结构，或者说简单结构)
2. Monolithic kernel(单一大内核)
3. Modular kernel(模块化结构) **←Linux的结构**
4. layered approach(层次结构)
5. Microkernel(微内核结构) **←鸿蒙OS的结构**

微内核结构的优势：可扩展性、可移植性、安全性和可靠性

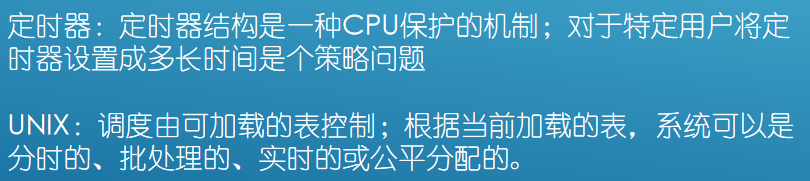
微内核结构的性能瓶颈：在于进程通信，采用message-passing(消息传递)

1. Hybrid structure(混合内核)
2. Exokernel(外核)

**·机制和策略相分离**

机制决定如何来做，策略决定做什么

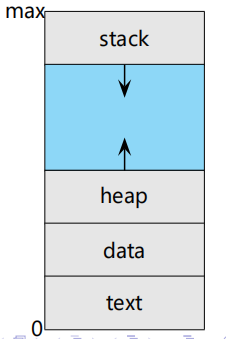
**经典例子：**timer(定时器)、priority(优先级)



1. **进程**

**§3 Process**

**·进程地址空间**



**·Program vs Process(进程)**

Program：**静态实体**

Process：**动态实体**

Process：A program in execution.

**·程序并发执行时的特征**

1. 间断性
2. 失去封闭性
3. 不可再现性

**·程序并发执行的条件(Bernstein’s conditions)**

必须防止不可再现性。

可再现性：只要程序执行时的环境和初始条件相同，都将获得相同的结果。

**·进程的五大特征**

1. **动态性----最基本的特性**

具有生命期：它由创建而产生，由调度而执行，因得不到资源而暂停执行，以及由撤销而消亡。

1. **并发性**

多道，既是进程也是OS的重要特征。

1. **独立性**

进程是一个能独立运行的基本单位，也是系统中独立获得资源和独立调度的基本单位。

1. **异步性**

进程按各自独立的、不可预知的速度向前推进。

导致“不可再现性”。

OS必须采取某种措施来保证各程序之间能协调运行。

1. **结构特征**

从结构上看，进程实体是由程序段、数据段及进程控制块三部分组成

进程映像 = 程序段 + 数据段 + 进程控制块

**·进程状态模型（要会画）**

**三状态模型**（三种最基本状态）

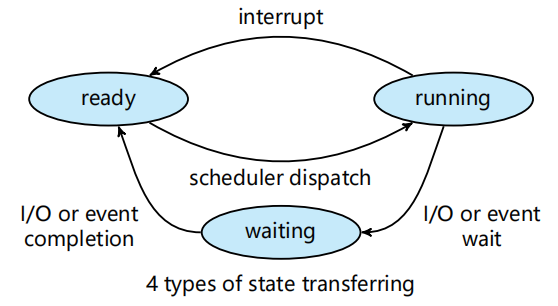
1. **ready** (就绪)：“万事具备，只欠CPU”

DataStructure: ready queue

1. **running** (执行)
2. **waiting** (等待, also blocked(阻塞), sleeping(睡眠))

DataStructure: waiting queue

进程等待事件发生：I/O完成、信号接收、资源分配...



**五状态模型**

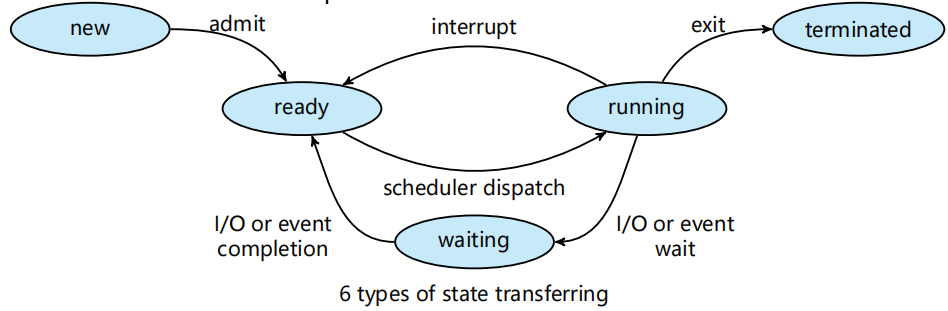
**new** (新状态)：进程被创建

初始化、资源预分配...

**terminated** (终止状态): 进程正常/异常完成执行

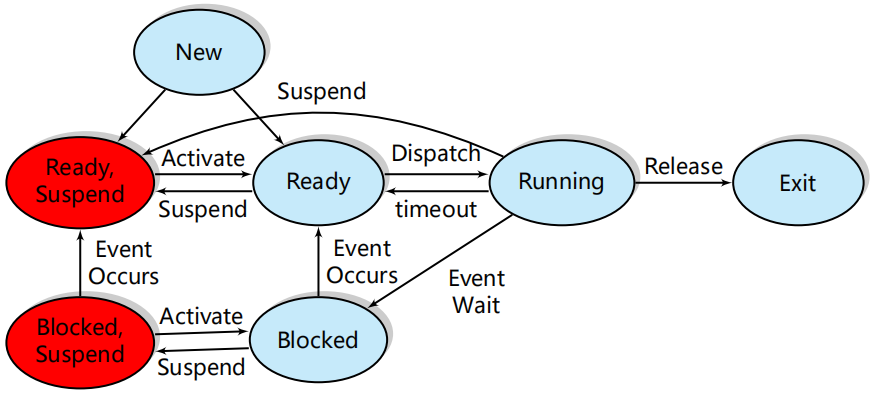
移出就绪队列，但仍未销毁

其它进程可能会从终止状态进程中取得信息



**七状态模型**

引入**挂起**



**·进程控制块，PCB**

**PCB是进程存在的唯一标志**

**·调度**

长期调度 (or job scheduler，作业调度)

决定哪些进程将进入就绪队列

短期调度 (or CPU scheduler，进程调度)

决定哪些进程将被执行，并分配CPU

**长期调度**控制**多道程序度（内存中进程的数量）。**

中期调度（内存调度）可以降低多道程序度

将暂时不能运行的进程调度至外存等待，当它们已具备运行状态且内存稍有空闲时，由中期调度决定将它们重新调入内存

短期调度不影响多道程序度

**·上下文切换** 保存旧进程的状态，加载新进程的状态

中断/调度时发生

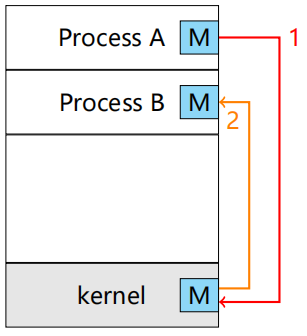
CPU寄存器、进程状态、内存管理信息等被保存在进程的PCB中

**·进程间通信，IPC**

**通信模型**

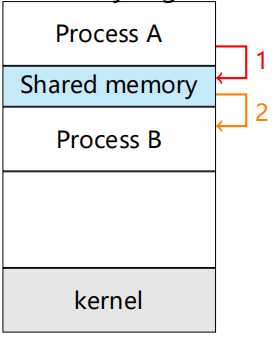
1. Message-passing model

适合少量信息传递，易于实现，通过系统调用交换信息



1. Shared-memory model

更快，仅在建立共享内存区时用到系统调用



**常见的进程间通信机制**

信号、 信号量、 管道、 消息队列、 套接字

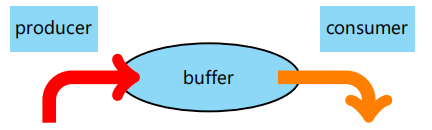
**·生产者消费者问题，PC问题**

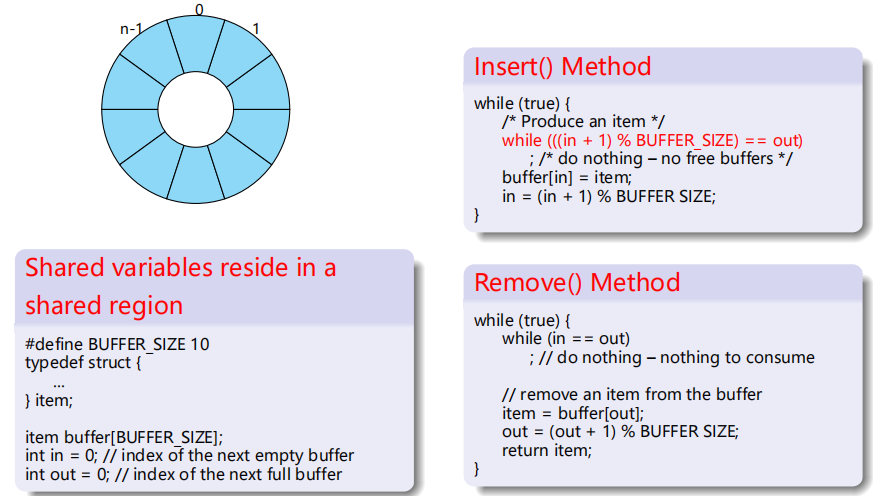
进程间合作的范式

生产者产出信息，消费者消耗信息

共享内存解决方案

一个生产者消费者共享的buffer





注意只有BUFFER\_SIZE - 1个元素可用（为了判断循环队列空还是满）。

**§4 CPU scheduling**

**·调度标准**

1. CPU utilization (CPU 利用率)

让CPU尽可能地保持忙碌

1. Throughput (吞吐率)

每个时间单位完成执行的进程数量

1. Turnaround time (周转时间)

从进程被提交开始至进程执行结束的时间

在等待进入内存、在就绪队列等待、在执行中的时间都要算

1. Waiting time (等待时间)

在就绪队列中等待的时间之和

1. Response time (响应时间)

从进程被提交开始至首次得到响应（开始执行）的时间

**·调度算法**

**Gantt Chart(甘特图)**

要会画

**FCFS**

护航效应/护卫效应，Convoy effect

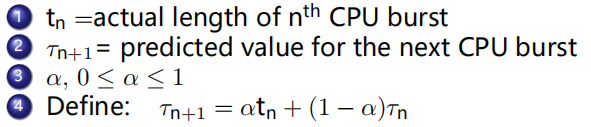
所有其他进程都在等待一个大进程使用完CPU

短进程排在长进程后

**SJF**

最优：平均等待时间最少

指数平均公式：预测下一个CPU Burst的长度



**Priority(优先级)**

通常规定小优先数代表高优先级

可能导致：饥饿

解决方法：Aging，随时间不断提高等待中进程的优先级

**优先级算法和SJF（是特殊的优先级算法）都可能导致饥饿现象！**

**RR（时间片轮转）**

通常平均周转时间更长但平均响应时间更短

**·多处理调度**

**矛盾：**亲和性（processor affinity）与 负载平衡（load balancing）

**亲和性：**进程在处理器之间迁移代价太大，进程对当前运行它的处理器有亲和性

**负载平衡：**尝试让各个处理器的工作负载保持平衡

**·实时调度**

优先级反转问题两种解决方案：优先级继承、优先级天花板

**§5 Threads**

**·线程**

线程是CPU利用的基本单元，包含线程ID、PC、寄存器集、栈

同一进程下的多线程可共享代码段、数据段（全局变量等）、**堆（但栈不可以，每个线程有独立的栈）**、其它OS资源等

区分线程与进程！不同**进程**间（如fork生成的父子进程）是不共享全局变量的！

**·优点**

**响应性、资源共享、经济、可伸缩性**

**·并行，Parallelism**

任务并行：将许多可以解决问题的任务分割，然后分布在一个或者多个核上进行程序的执行。

数据并行：将可以解决问题的数据进行分割，将分割好的数据放在一个或者多个核上进行执行；每一个核对这些数据都进行类似的操作。

**§6 Process synchronization**

**·竞争条件**

多个进程并发访问与处理同一数据，且执行结果依赖于访问数据的顺序。

**·临界区问题**

临界区问题的解决方案满足三个条件：

1. 互斥
2. 空闲让进
3. 有限等待

**·信号量，Semaphore**

**整型信号量**

初始化+两个可修改S的标准操作（wait & signal）



作用：

1. Counting semaphore（Resource semaphore）

记录可使用资源的数量，初始化为N

资源请求：wait

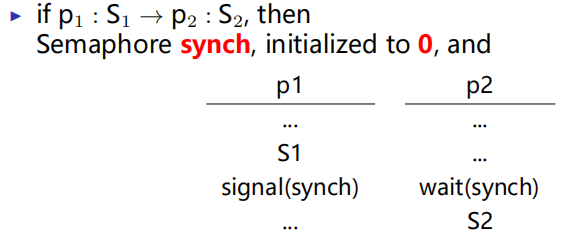
资源释放：signal

1. Binary semaphore

值为0或1，可以用N=1的Counting semaphore实现

1. 解决同步问题的工具

例：描述前趋关系



缺点：

造成忙等（busy waiting）

忙等：是一种进程执行状态。进程执行到一段循环程序的时候，由于循环判断条件不能满足而导致处理器反复循环，处于繁忙状态，该进程虽然繁忙但无法前进。

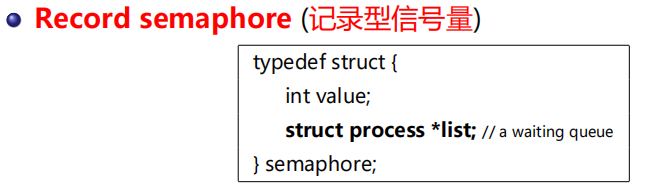
这种信号量也称为自旋锁（spinlock）

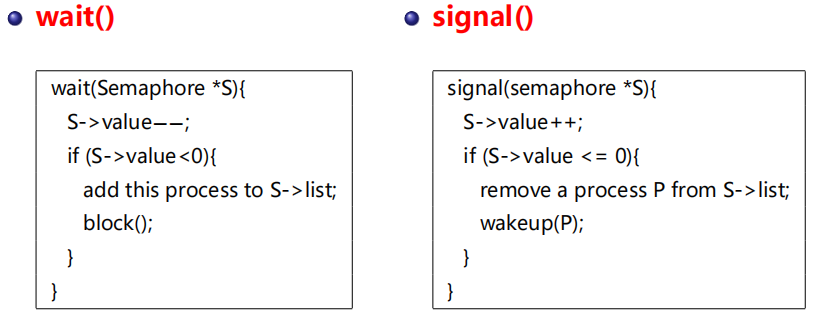
合适的使用情况：

1. busy waiting (for I/O) time < context switching time
2. multiprocessor systems & busy waiting time is very short

为何使用自旋锁：在多处理环境下使用自旋锁可以节省线程状态切换带来的开销

**记录型信号量，Record semaphore** ←no busy waiting





分析

对于wait操作：

当value≥1时，说明有资源剩余；申请资源只需要减1

当value＜1时，说明没有资源剩余；此时，减去1，并等待

对于signal操作：

若value≥0，说明没有等待者，不必唤醒，只需加1释放资源

若value＜0，说明有等待者；加1缩短等待队列长度，并唤醒1个进程

（资源分配给这个进程）

查看value：

value≥0，说明没有等待者，此时，value值表示剩余资源的个数

value＜0，说明有等待者，此时L上有等待进程；此时，value的绝对值表示等待进程的个数

**信号量误用问题**

1. 死锁Deadlock

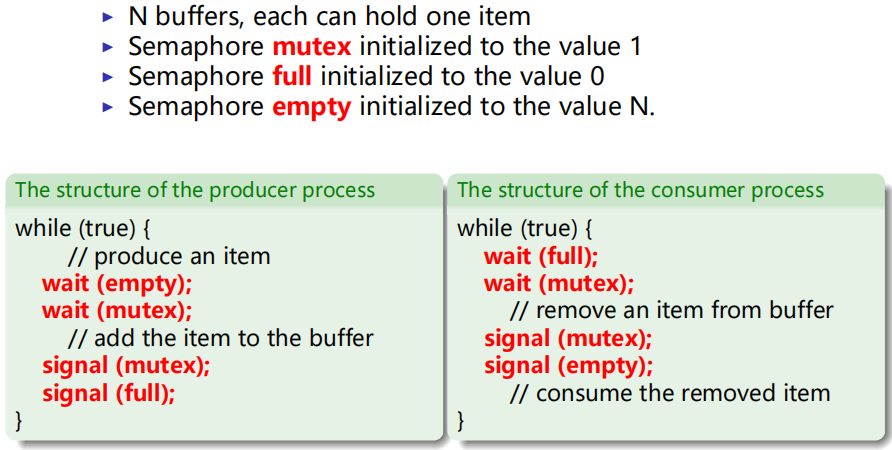
两个或多个进程无限期地等待一个事件，而这个事件可能是由其中一个等待进程引起的

1. 饥饿Starvation

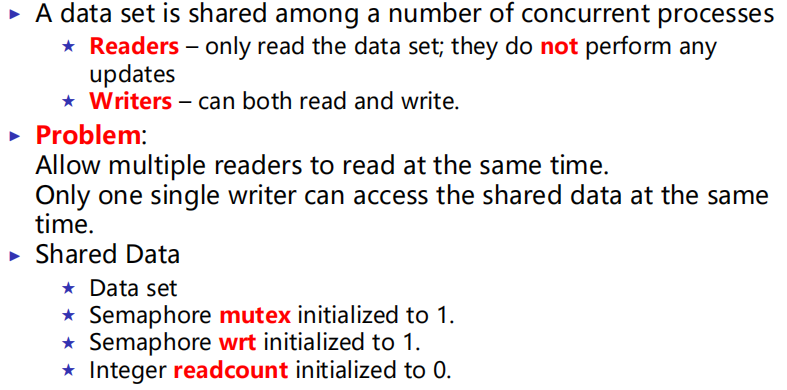
无限阻塞，进程不能从挂起它的信号量队列中移除。

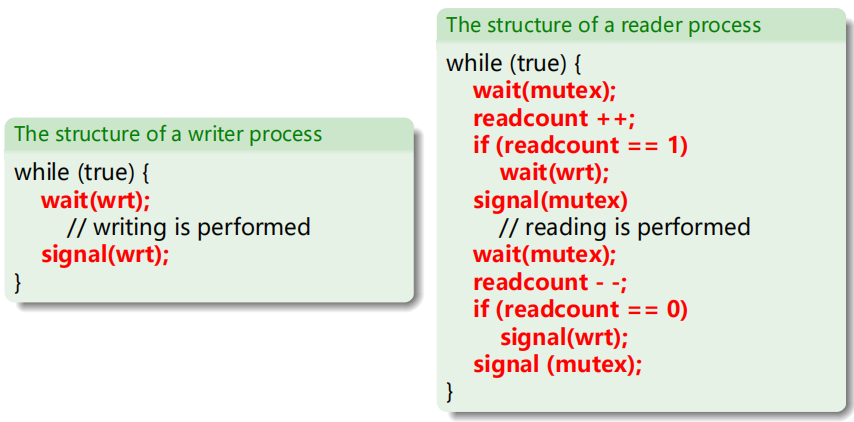
**·经典同步问题**

**生产者-消费者问题，Bounded-Buffer Problem**

****

**读者-写者问题，Readers and Writers Problem**

****

****

该方案尚存在问题，分析：

**a) 请举例说明什么情况下会产生写者的 starvation 现象**

有读者在占用资源且不断有新的读者时可能导致写者的饥饿现象

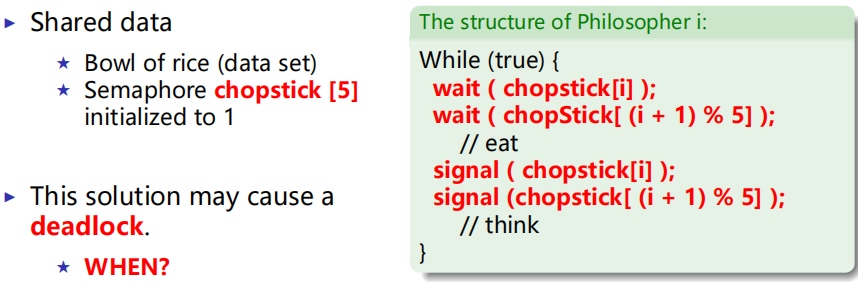
**b) 请举例说明什么情况下会产生读者的 starvation 现象**

如果信号量的等待队列是FIFO的，读者不会饿死

**c) 解决上述 starvation 现象 ←**限定读者容量

读者进程：起始readcount == N时，释放mutex然后直接continue/ break

**哲学家就餐问题，Dining-Philosophers Problem**

****

该方案尚存在问题，分析：

**a)请举例说明什么情况下会产生 deadlock 现象**

五个人同时拿起了左(右)手边的筷子(筷子顺逆时针排列)

**b) 请举例说明什么情况下产生 starvation 现象**

在上述情况a)下就可以导致饥饿（死锁也可以是饥饿）

**c) 解决方案 ←**限定同时就餐人数（万能？）

允许至多四个哲学家共同就餐，或：

奇数编号先拿左边筷子再拿右边筷子，偶数编号先拿右边筷子再拿左边筷子

**§7 Deadlock**

**·死锁**

一组阻塞的进程，每个进程持有一个资源，并等待获取集合中另一个进程持有的资源。

**·死锁判定的四个必要条件**

1. 互斥

同一时间一个资源只有一个进程可以使用。

1. 持有并等待

持有至少一个资源的进程正在等待获得其他进程持有的资源。

1. 不剥夺

只有在持有资源的进程完成其任务后，该进程才能主动释放资源。

1. 循环等待

存在一个等待进程集合{P0, P1，…， Pn}，其中P0等待P1持有的资源，P1等待P2持有的资源，…，Pn−1等待Pn持有的资源，Pn等待P0持有的资源。

**·死锁预防**

分别从死锁判定的 4 个必要条件加以考虑：

1. 互斥

无

1. 持有并等待

进程开始执行前请求并分配所有资源或者当进程没有资源时才能请求资源。

1. 不剥夺

可以主动剥夺进程的资源。

1. 循环等待

对所有资源进行排序编号，要求进程按照编号递增的顺序进行申请资源。在申请较低编号的资源之前可先释放较高编号的资源。

**·死锁避免**

**安全状态**

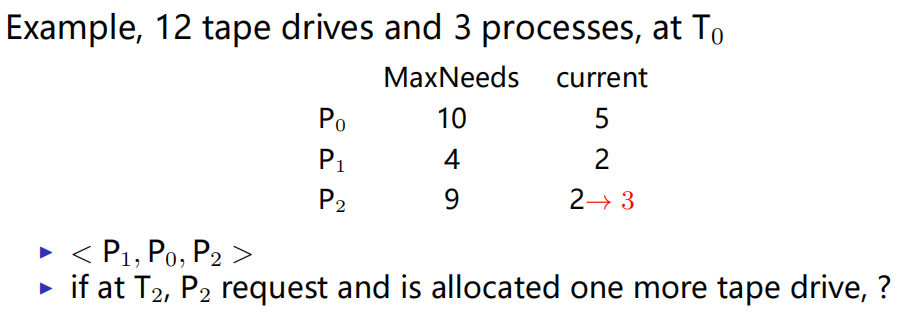
当进程请求可用资源时，系统必须决定立即分配是否使系统处于安全状态。

如果存在一个包含系统中所有进程的安全序列< P1, P2, ..., Pn >，

使得对于每个Pi, Pi仍然可能请求的资源可以由当前可用的资源 + 所有j < i的Pj持有的资源来满足，则系统处于安全状态。

若系统处于安全状态则无死锁，反之则可能出现（但不一定）死锁。

死锁避免：保证系统永远处于安全状态。



若P2 current 2→3，则不存在安全序列！

**银行家算法，Banker’s Algorithm**

1. 数据结构

**n**  //进程数

**m**  //资源种类数

**Available[m]**  //各类资源可用量

**Max[n][m]**  //进程对资源的最大可能申请量

**Allocation[n][m]**  //进程已被分配资源量

**Need[n][m]**  //进程还可能需要的资源量（Need = Max - Allocation）

1. 安全算法，Safety Algorithm

**Work[m]** //当前资源可用量

**Finish[n]** //进程完成标志数组

①初始化：令Work = Allocation，Finish[i] = false (i=0,1,...,n-1)

②寻找i满足：Finish[i] = false 且 Need[i] ≤ Work。若i不存在，转④

③Work = Work + Allocation[i]，Finish[i] = true，转②

④若对所有的i满足Finish[i] = true，则系统处于安全状态

1. 资源请求算法，Resource-Request Algorithm

**Request[m]** //某一进程Pi请求的资源量

①判断Request ≤ Need[i]，否则错误

②判断Request ≤ Available，否则资源不足，Pi需等待

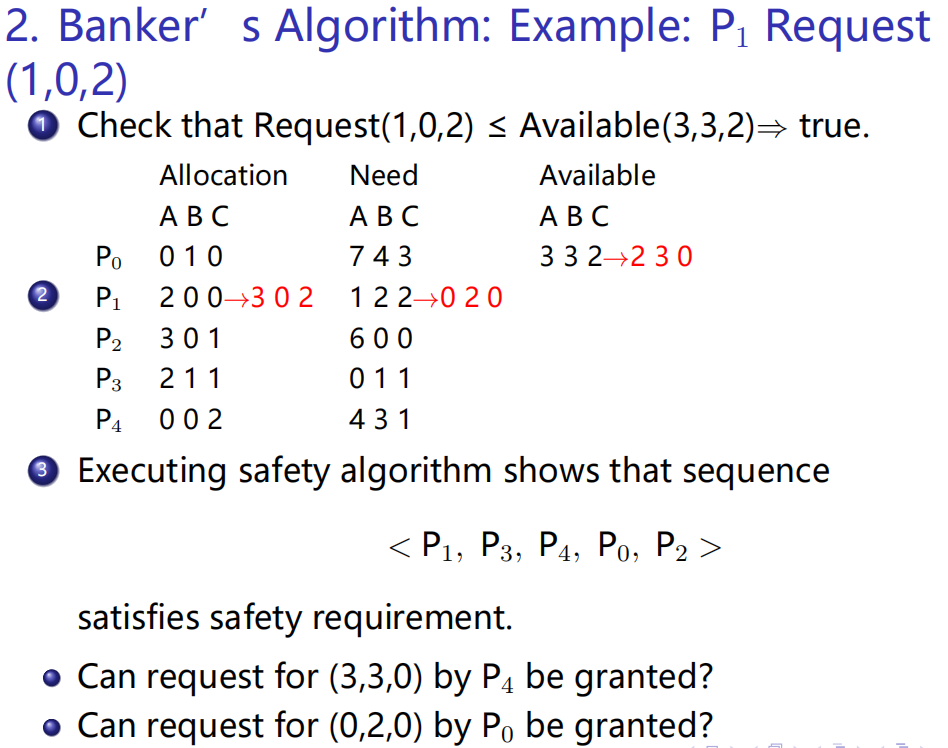
③向Pi分配资源：

Available = Available - Request

Allocation[i] = Allocation[i] + Request

Need[i] = Need[i] - Request

实例



①不能，分配之后不是安全状态，会进入死锁。

②可以，分配后可能的安全序列：

(P3 , P1 , P0 , P4 , P2 )

(P3 , P1 , P0 , P2 , P4 )

(P3 , P1 , P2 , P0 , P4 )

**·死锁检测**

**检测算法** ←与安全算法类似

**Work[m]** //当前资源可用量

**Finish[n]** //进程完成标志数组

①初始化：令Work = Allocation，

若Allocation[i] ≠ 0，则Finish[i] = false ；否则Finish[i] = true (i=0,1,...,n-1)

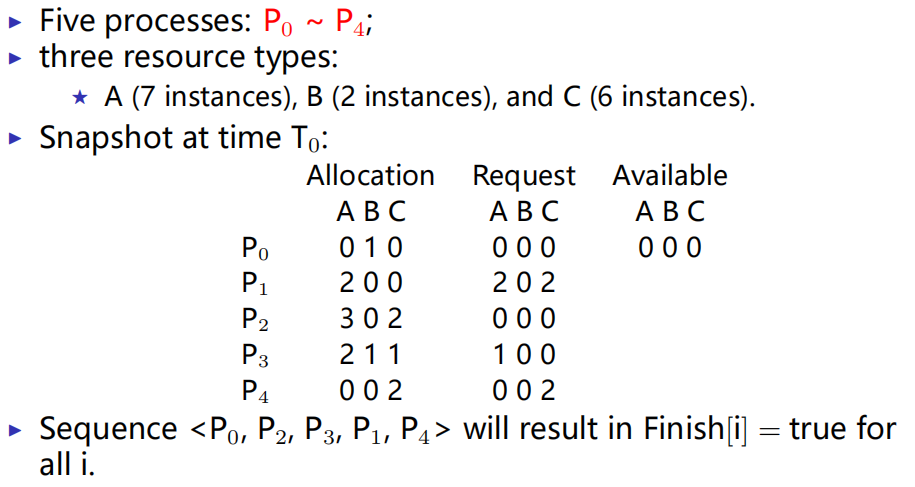
②寻找i满足：Finish[i] = false 且 Request[i] ≤ Work。若i不存在，转④

③Work = Work + Allocation[i]，Finish[i] = true，转②

④若存在i满足Finish[i] = false，则系统处于死锁状态。

更进一步地，若Finish[i] = false，则Pi处于死锁中。

实例



**·死锁恢复**

1. 中止所有处于死锁中的进程 （显然不如2好）
2. **一次中止一个进程，直到死锁消失** —— 如何最小化成本？

需要解决三个问题：

1. 选择受害者(victim)——最小化成本。
2. 回滚——返回到安全状态，重新启动进程。
3. 饥饿——相同进程可能总被选为受害者，故要在成本因素中考虑回滚次数。
4. **内存**

**§8 Main Memory**

**·地址**

物理地址（绝对地址）：内存单元的真正地址

逻辑地址（虚拟地址）：CPU所生成的地址

**·地址绑定（时机）**

编译时、装入时、运行时

**·MMU（Memory-Management Unit）**

将虚拟地址映射到物理地址的硬件设备

**·程序加载，Program loading**

1. 绝对装入
2. 可重定位装入
3. 动态运行时装入

**·程序链接，Program linking**

1. 静态链接
2. 装入时动态链接
3. 运行时动态链接

**·连续内存分配**

**单一连续分配**

**多分区分配**

多道程序度由分区数量约束

碎片

**内部碎片：**分配内存空间大于需求空间，分区内部存在未被利用的空间

**外部碎片：**总内存空间随着进程的换入换出，分区之间可能出现的小内存块

**消除外部碎片的方法：**紧凑法（Compaction），即操作系统不时地对进程进行移动和整理。这项技术需要动态重定位寄存器的支持。

1. 固定分区

**缺点：**内存利用率差，碎片多

1. 动态分区

**分配：**

First-fit（首次适应）：分配首个足够大的hole

Next-fit（循环首次适应）：分配第二个足够大的hole

Best-fit（最佳适应）：分配足够大的hole中最小的

Worst-fit（最差适应）：分配最大的hole

**回收：**要判断前后相邻块是否需要合并

**缺点：**随着分配进行，空闲分区分散在内存各处；尽管有回收，但内存仍越划分越碎，形成大量外部碎片

**·离散内存分配**

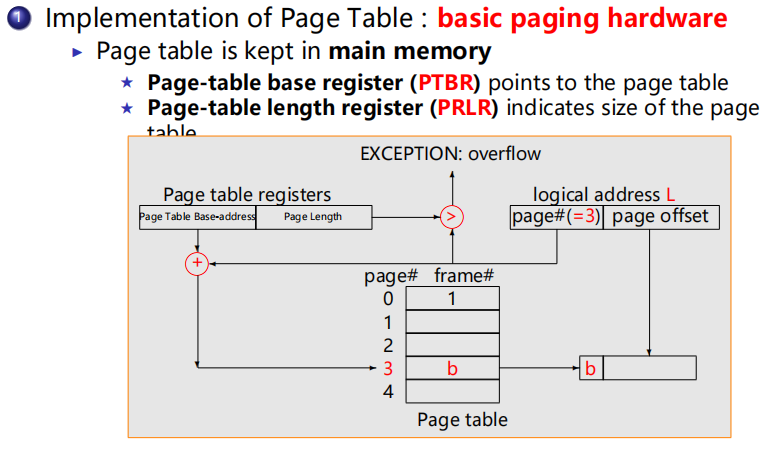
**分页**

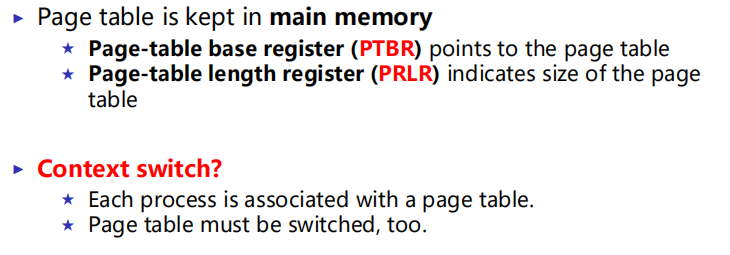
将物理内存分成固定大小的块，称为frames（物理页框）

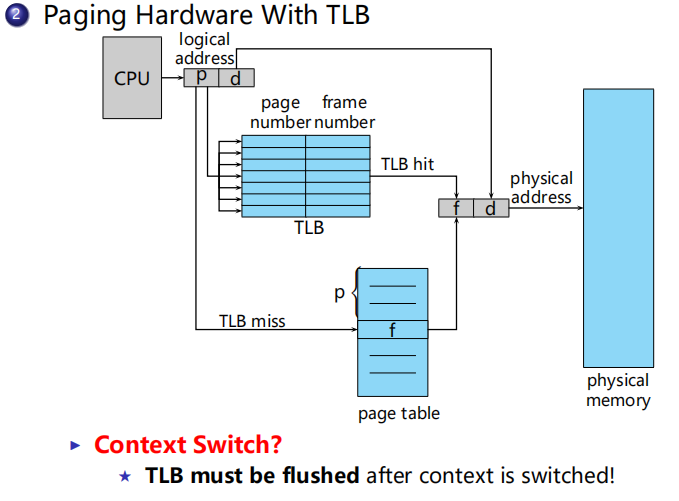
将虚拟内存分成同等大小的块，称为pages（逻辑页，页）

内部碎片大小 ＜ page size

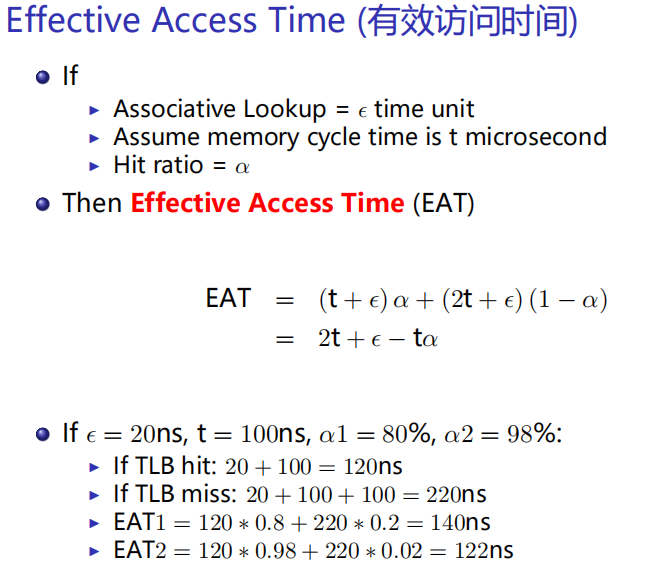
**Paging hardware ←**要能画有无TLB的两种图



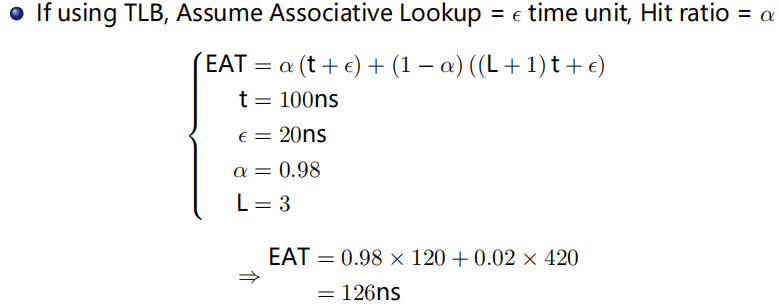




**EAT**



多级页表EAT（L为页表级数）



**内存保护**

给页表中每一项添加valid位

**§9 Virtual Memory**

**·局部性原理**

时间局部性：如果程序中的某条指令一旦执行，则不久之后该指令可能再次被执行；如果某数据访问，则不久之后该数据可能再次被访问。

空间局部性：一旦程序访问了某个存储单元，则不久之后其附近的存储单元也可能被访问。

**·虚存的特征**

1. 多次性 ←最重要特征

一个作业被分成多次装入内存执行

1. 对换性

允许在进程运行过程中部分换入换出

1. 虚拟性

是逻辑上的补充

**·虚存的精髓**

部分装入、按需装入、置换

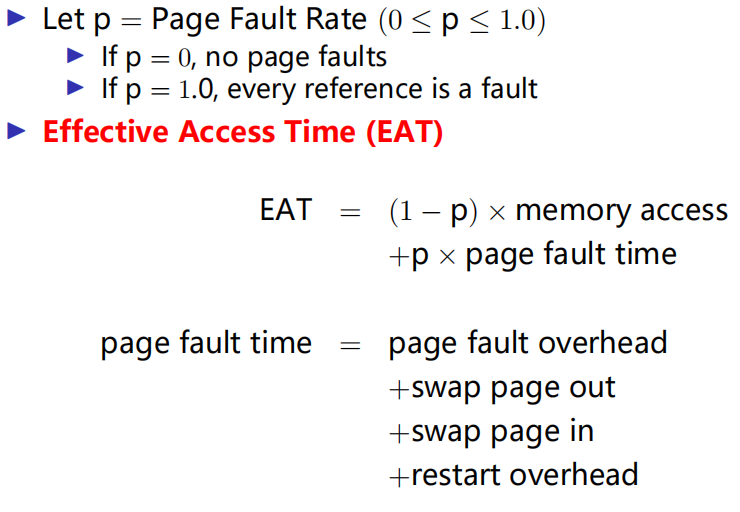
**·按需调页，Demand Paging**

仅在需要时才将页面调入内存

简要过程：

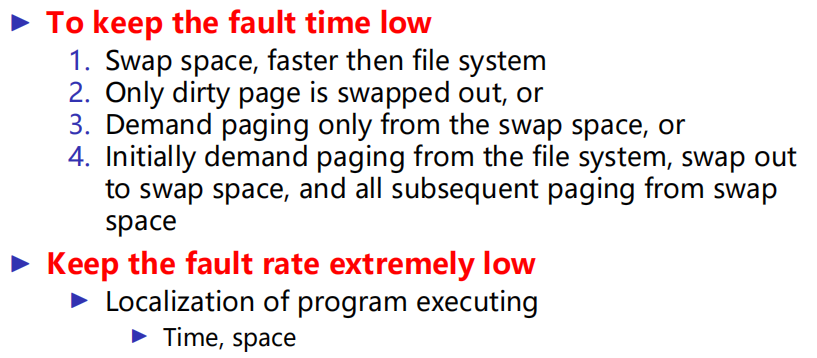
在请求分页系统中，只要求将当前需要的一部分页面装入内存。每当所要访问的页面不在内存中时，便产生一个缺页异常，请求操作系统将所缺的页面调入内存。此时应将缺页的进程阻塞（调页完成唤醒），若内存中有空闲块，则分配一个块，将所缺页面调入内存；若内存中没有空闲块，则要按页面置换算法淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存）

按需调页性能计算



提升按需调页性能的两个要点

减少**缺页中断时间**、降低**页错误率**



通过时间、空间局部性原理降低**页错误率**

通过交换空间的使用等减少**缺页中断时间**

**·页面置换算法**

**FIFO**

贝莱迪异常现象：某些时候，物理页框越多，缺页异常反而越多

**OPT**

最低缺页异常率

**LRU**

**·物理页框分配**

ISA决定了应给进程分配的最小物理页框数

**·抖动**

进程频繁地进行页面的换入换出

**根本原因：**不合理的多道程序度

应减少多道程序度，使进程得到更多物理页框

1. **文件系统和IO**

**§10 FileSystem**

**·什么是文件**

文件是一个逻辑存储单元，是辅存中的相关信息的一个命名集合，是位、字节、行或记录的一串序列。

**文件属性**

名称、标识符、类型、位置、大小、访问控制信息、时间、日期、用户标识等

这些文件属性信息保存在目录结构（directory structure）中，目录结构也被保存在辅存中。

**文件操作**

Create、Write、Read、Seek、Delete、Truncate...

打开文件列表、文件锁...

**文件类型**

exe、obj、c、txt、mp3...

**·访问方式**

顺序访问、随机访问、索引访问

**·目录和目录结构**

**目录（Directory）：**包含分区内所有文件信息的节点集合

**目录结构：**单层目录、双层目录、树形结构目录、无环图目录、通用图目录

**§11 FS Implementation**

**·目录实现** 线性、哈希

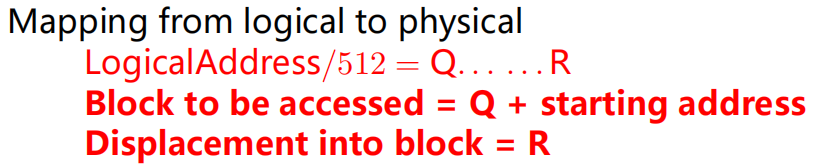
**·分配方法**

**连续分配**

每个文件占用磁盘上一组连续的块

目录项只需开始位置（块号）、长度（块数）

映射方式



优点：支持顺序访问和随机访问

缺点：碎片，浪费空间；文件不能增长，或文件大小需提前得知。

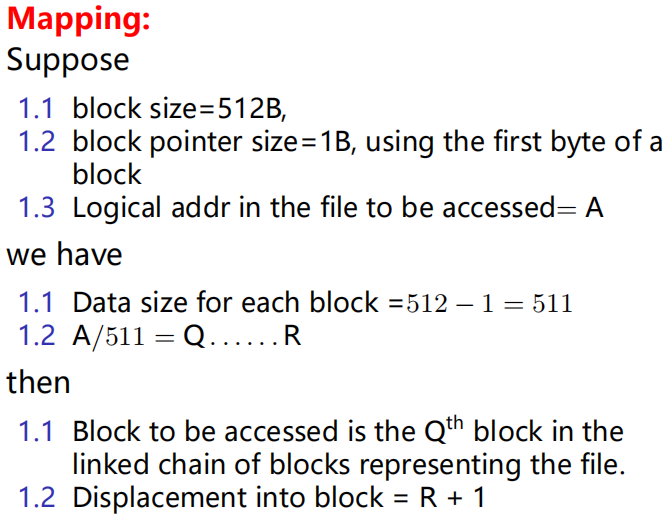
**链接分配**

每个文件是磁盘块的链表：块可以分散在磁盘上的任何地方

1. 隐式链接

每个目录项有指向文件首个及末尾块的指针，每个块有指向下一个块的指针。

映射方式



优点：简单，不浪费空间（按需分配）

缺点：不支持随机访问，且指针需要占用内存空间

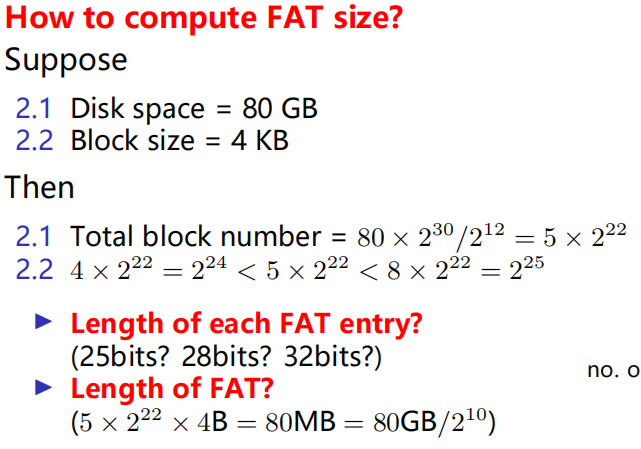
1. 显式链接

**File Allocation table, FAT**

在每个分区的开始部分留出一部分用于保存FAT，每个磁盘块在FAT中都有一个入口，包含了：

下一个块的索引、end-of-file标识（对最后一个块）、0（对未用到的块）。

目录项有首个块号

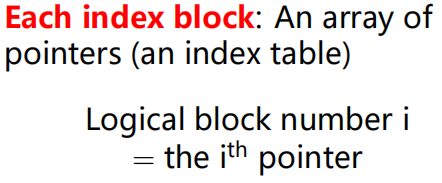


优点：支持随机访问

缺点：不是很有效率

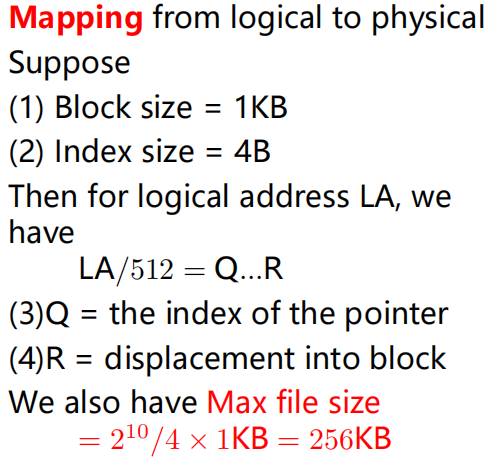
**索引分配**

将所有指针集中到一个位置——索引块，每个文件都有自己的索引块



目录项包含索引块地址

映射方式



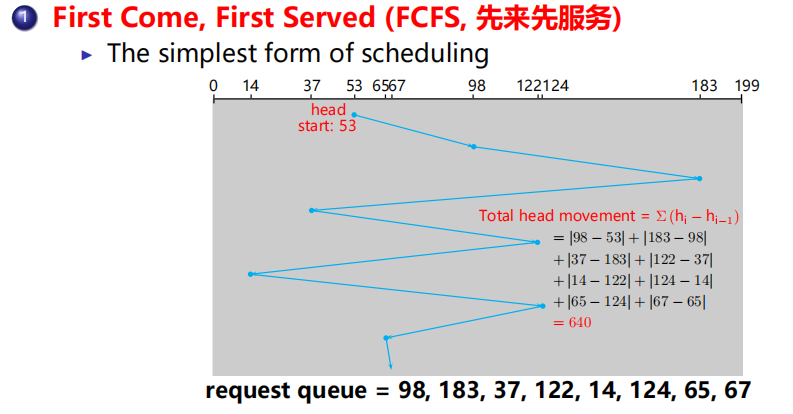
优点：随机访问、没有外部碎片的动态访问

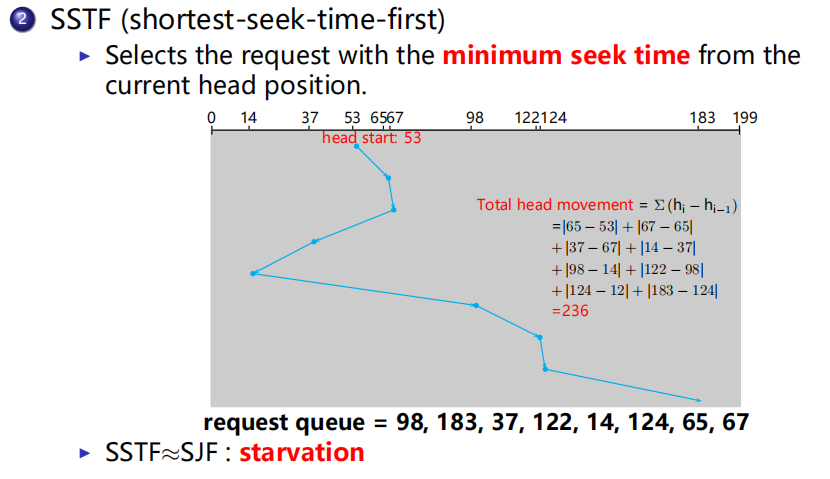
缺点：索引块有开销，文件大小有限制（单个索引块包含指针数有限）

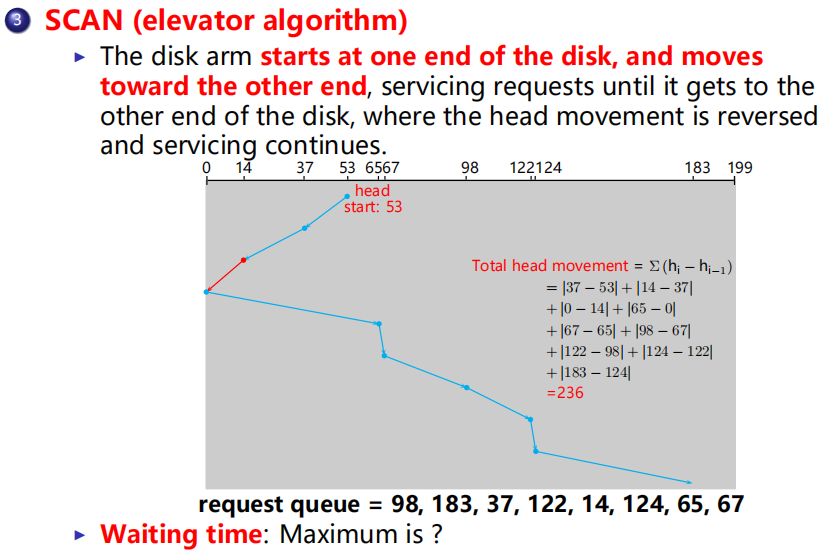
**组合方式**

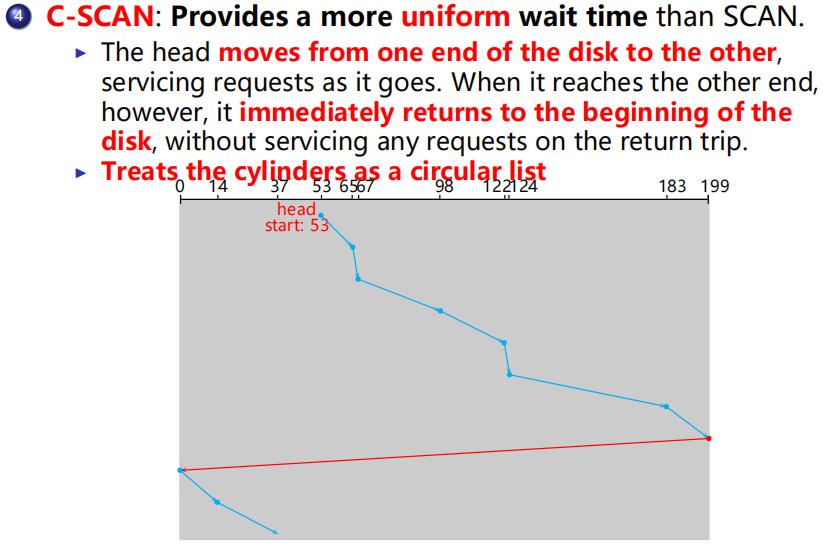
**§12 Mass-Storage structure**

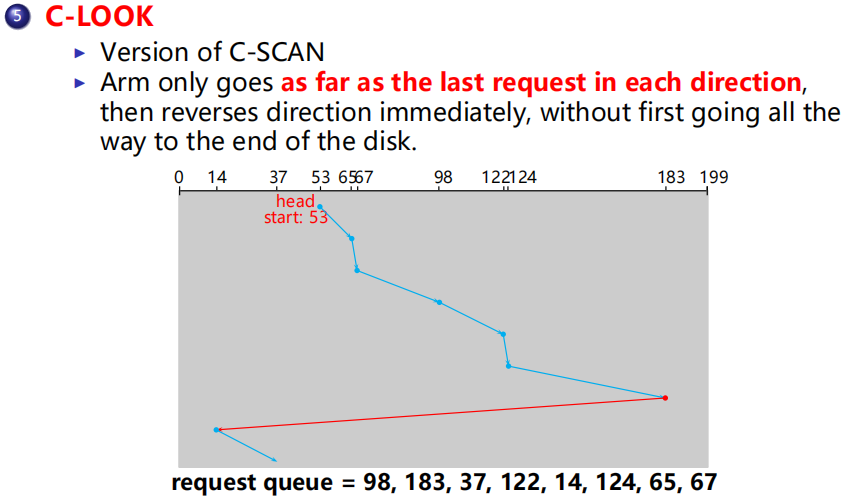
**·磁盘调度**

****

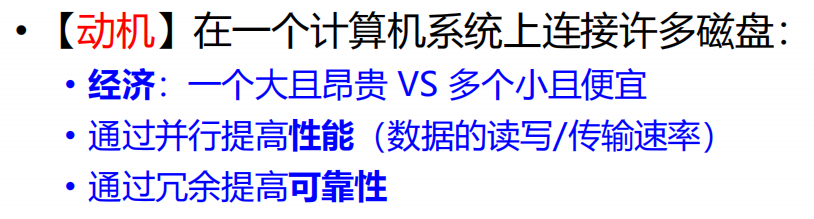
****

****

****

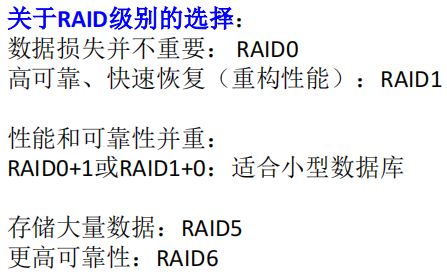
****

**·磁盘冗余阵列，RAID**









**§13 IO**

**·IO硬件和IO控制方式**

轮询、中断、DMA

**·IO子系统功能**

IO调度、缓冲、高速缓存、Spooling、IO保护

**看不完辣，不写了**