**概述**

**OS**：is a set of software for managing computer resources.

**特权指令**：**1.**允许和禁止中断,控制中断禁止屏蔽位**2.**切换处理进程**3.**存取用于主存保护的寄存器**4.**执行I/O操作**5.**停止一个中央处理器的工作**6.**清理内存**7.**设置时钟**8.**建立存储键**9.**加载PSW(程序状态字,用于管态和目态切换)

**OS接口**：命令与程序接口

**系统调用**：用户调用函数->调用函数库函数->调用内核函数.系统调用实现在内核完成,用户态函数在函数库实现.

**↑参数传递**：**1.**寄存器传递**2.**块,表,内存中(块地址通过寄存器传输）**3.**堆栈存储（后两者不限制传递参数的数量和长度

**系统调用类型**：**1.**进程控制**2.**文件管理**3.**设备管理**4.**信息维护**5.**通信**6.**安全

**ABI**：Application Binary Interface,应用程序二进制接口,定义二进制代码的不同组件如何与OS对接

**操作系统结构**：**1.**Simple structure简单结构,e.g. MS-DOS **2.**Layered Approach 层次化结构, layer0 (hardware)-layerN(user interface),每个操作执行选择适合自己的最底层layer **3.**Monolithic Kernels Structure单/宏内核结构,构建OS内核传统方法,内核全部代码打包到一个文件,内核每个函数可访问内核其他部分,e.g. UNIX, OS/360, Linux. 组件直接通讯,高效;debug困难 **4.**Microkernel微内核,只有最基本功能由内核实现,其余委托给独立进程,通过接口与内核通信, e.g. Windows NT,…,8,10, Mac OS. 易拓展移植,内核运行代码少可靠安全;通信开销大 **5.**Modules模块, 面向对象,核心组件独立,通过接口与其他对话,可根据需要在内核中加载,与layer类似但更灵活,e.g. Linux, Solaries **6.**Hybrid Systems混合结构 **7.**Virtual Machines虚拟机

**三大基本操作系统**：**1.** batch system批处理系统,适用计算量大,不需要与用户交互的大型作业,提高了CPU利用率.资源利用率高,吞吐量大;平均周转时间长,无交互能力**2.** Time-sharing system分时系统,交互性强的多用户系统. 人机交互,共享主机.e.g. Linux **3.** Real-time system实时系统,事件驱动,较少人为干预,可靠性强,交互性弱于分时系统

**进程**

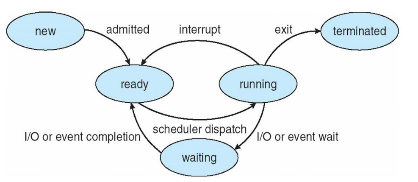
**进程**：正在执行的程序

\*资源拥有单位,与CPU任务相关联

\*调度单位,即线程

组成有1.program code,代码段2. PC, program counter 3.registers寄存器 4. Data section数据段(global data) 5. Stack(temporary data) 6. Heap (dynamically allocated memory)

**进程状态**：1.New, 进程被创建 2.Running, 指令正被执行 3.Ready, 等待CPU分配 4.Waiting(等待、阻塞), 进程等待event发生 5.Terminated, 进程结束执行



**状态改变事件**：1.Program action(system call) 2.OS action (scheduling decision) 3.External action(interrupts)

**Process vs. Program**：动态与静态,暂时与永久,组成不同;多次执行,一个程序对应多个进程;调用关系,一个进程包括多个程序

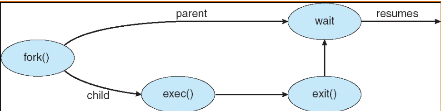
**PCB(Process Control Block)**：进程控制块,包括: 1.Process state 2.Program counter 3.CPU registers 4.CPU scheduling information 5.Memory-management information 6.Accounting information 7.File management 8.I/O status information

**进程调度队列**：**1.** Job queue, 在系统中的进程队列**2.** Ready queue, 存在于内存中,等待执行**3.** Devide queue, 等待I/O设备

**Schedulers**：**1.**Long-term scheduler (or Job scheduler), 长程调度(或作业调度),选择进程进入ready queue, 现代操作系统很少用了 **2.**Short-term scheduler(or CPU scheduler), 短程调度(或CPU调度),选择进程被执行或分配分配CPU,执行频繁,要求速度快**3.**Medium-term scheduler中程调度 **\*\*\*1.** I/O-bound process, I/O进程 **2.**CPU-bound process,CPU进程

**Context Switch上下文切换**：CPU切换进程,通过上下文切换来存储旧进程状态并加载新进程状态

**PID**：process identifier,进程标识符

**子父进程**：父进程与子进程同时执行,子进程终止前,父进程一直等待

调用pid=fork(),子进程返回值为0,父进程返回值为子进程的PID.两个进程具有完全一样的用户级上下文.

**进程通讯**

**model**：**1.**共享memory

**2.**message passing消息传递,消息放在message queue. Blocking和 non-blocking,同步和异步

**类型**：

**1.**直接通信,send(P,message)给P发消息和receive(Q,message) 从Q获得消息,link自动建立,一个链接对应一对进程通讯,链接可能单向,大部分情况是双向的

**2.**间接通信,send(A,message)和receive(A,message)从中介A收发消息.共享端口,一个链接可能跟多个进程相关,每对进程可共享多个通信链接,可单向或双向,允许链接最多与两个进程相关联,同一时间点,只允许一个进程进行receive,允许OS随机选择接收者,发送者被推送谁收到了消息

**通信机制**：**1.**信号signal **2.**共享存储区shared memory **3.**管道pipe(Ordinary pipes匿名管道, 子父进程,单向;Named pipes命名管道,不存在子父关系) **4.**消息message **5.**套接字,socket

**线程和并发**

**线程Thread**：**1.**可定义为进程内一个执行单元或一个可调度实体**2.**有执行状态(状态转换) **3.**不运行时保存上下文**4.**有执行栈,一些局部变量的静态存储**5.**可存取所在进程的内存和其他资源**6.**可以创建、撤销另一个线程

**特点**：**1.**不拥有系统资源**2.**多个线程可并发执行**3.**开销小、切换快

**属于统一进程的线程共享**：**1.**code section **2.**data section **3.**operating-system resources(共享进程的虚拟地址空间). Heap memory and global variables, but has separate register values and stack.

**Pros**：**1.**创建,切换,结束花费时间少 **2.**统一进程内线程共享内存和文件,故通信无需调用内核**3.**适合多处理机系统（不需要顺序执行的时候,并且阻塞时还可以切换）

**User threads用户级线程**：不依赖于OS核心; 一个线程阻塞,则整个进程在等待（一对多模型中）

**Kernel threads 内核级线程**：依赖于OS核心; 一个线程阻塞,不会影响其他线程

**多线程模型**：**1.**Many-to-one,多个用户级线程对应一个内核级线程, 一个线程阻塞,整个进程阻塞 **2.**One-to-one, 每个用户级线程都映射到内核级线程,每个内核级线程都可在多处理器上并行运行,阻塞可调度其他线程 **3.**Many-to-Many, 允许多个用户级线程映射到多个内核级线程 **4.**Two-level Model,总和2和3

**CPU调度**

**When to make CPU scheduling**：1.Running->waiting state

2.Running -> ready state

3.Waiting-> ready state 4.terminate

**How to**: 1. Nonpreemptive非抢占式 2.Preemptive抢占式

**Dispatch latency 调度延迟**：

running调度程序停止一个进程到启动一个进程所需要的时间

**调度准则**：**1.**面向用户

**\*周转时间Turnaround time** = termination time – arrival time, 完成时间-提交时间.

**\*等待时间Waiting time** = turnaround time – burst time

**\*平均周转时间**=

**\*加权周转时间W** = 周转时间/CPU执行时间

**\*平均带权周转时间**=

**\*响应时间Response time**=首次被相应事件(而不是输出结果时间)-进程提出请求时间

**\*截止时间**=开始截止和完成截止时间——实时系统

**2.**面向系统

**\*吞吐量Throughput**：单位时间完成进程数,不是平均周转时间的倒数(考虑并行)

**调度算法：**

**1.FCFS**,first come first serve

按照进程或作业提交顺序形成就绪状态的先后次序分派CPU,非抢占方式.有利于长进程和CPU Bound进程.不利于短进程和I/O Bound进程.最简单的算法

**2.SJF**,shortest-job-first,短进程优先算法.减少平均周转时间.非抢占式.

抢占式: 新进程到达时burst length小于当前进程剩余时间,则抢占.也称为**SRTF**(shortest remaining time first),最短剩余时间优先

SJF是最优的,它给一组进程提供了最短平均等待时间

**SJF变型：\*SRF**(shortest remaining time)最短剩余时间优先

**\*HRRN**(Highest response ratio next)最高响应比优先.响应比**R**=(等待时间+要求执行时间)/要求执行时间.FCFS和SJF的折中.

**3.Priority scheduling按优先级调度**,静态优先权和动态优先权.一般来说,最小整数=最高优先级.抢占与非抢占.

**4.Round robin(RR)时间片轮转**,解决饥饿.进程按FCFS排成一个队列,首进程执行一个时间片,用完后送到队尾.通过上下文切换执行新的首进程.进程可以未执行完一个时间片就让出CPU(如阻塞).

响应时间=进程数\*时间片

时间片长=>FCFS,时间片短=>context switch开销大.

**5.Multilevel Queue Scheduling多级队列调度**,将就绪队列分为若干个子队列,根据进程性质采用不同调度策略.e.g.前台(交互式)和后台(批处理)队列

**6.Multilevel Feedback Queue Scheduling多级反馈队列**,设置多个就绪队列,设置多个优先级,每个队列的时间片长度不同,优先级越低时间片长度越长. 新进程进入后,先放入队列1末尾,按FCFS调度.若队列1一个时间片未能执行完,则投入队列2末尾,按FCFS调度,以此类推. 仅当高优先级队列为空时,才调度低优先级队列的进程执行.若进程执行时有新进程进入高优先级队列,则抢占执行.被抢占进程投入原队列末尾.

**7.Multiple-Processor Scheduling多处理器调度**,更复杂,需要具有多处理器的同构处理器,非对称多处理(Asymmetric multiprocessing)

**线程调度**：1.Local Scheduling:线程库决定 2.Global scheduling:内核决定

**进程同步**

**进程最基本特性**：并行性、独立性

**临界区问题The Critical-Section Problem**：n个进程竞争使用同一些共享数据,当一个进程在其临界区执行时,不允许其他进程在其临界区执行.

**临界资源Critical-Section**：一次只允许一个进程使用（访问）的资源,e.g. 硬件打印机,软件的消息缓冲队列,变量,数组,缓冲区等

**临界区**：访问临界资源的那段代码

General structure of process P\_i

do{

entry section //进入区

critical section //临界区

exit section //退出区

reminder section //剩余区

}while(1);

**解决临界区问题的条件**：1.Multual Exclusion互斥 2.Progress空闲让进 3.Bounded Waiting有限等待

**Peterson’s Solution(解决两个进程的临界区问题)**：

P0：do{

Flag[0]=true;

Turn = 1;

While(flag[1] and turn = 1)

Critical section;

Flag[0]=false;

Remainder section

}while(1);

P1：do{

Flag[1]=true;

Turn=0;

While(flag[0] and turn=0)

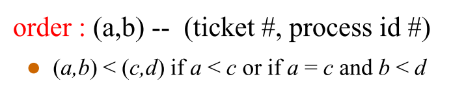
Critical section

Flag[1]=false;

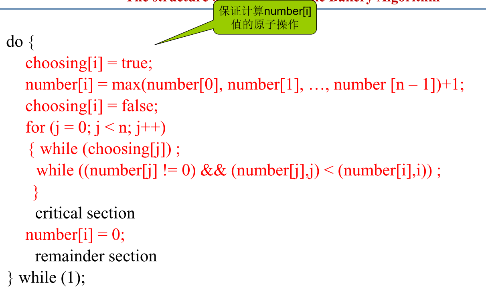
Remainder section

}while(1);

**面包房算法Bakery algorithm(解决n个进程的临界区问题)**：进入临界区前,进程获得一个编号,持有最小编号的进程进入临界区.编号相同时,i和j中小的那个先执行.生成的编号总是升序或相等的.

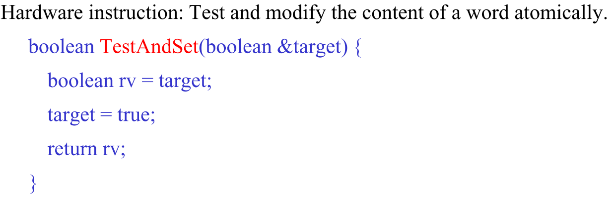


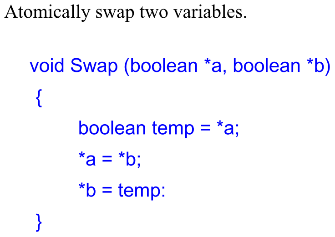
Choosing[i]:为真表示正在获取排队号; number[i]:值为排队号,为0表示未参加排队.



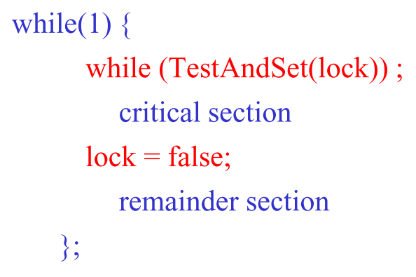
**硬件同步Synchronization hardware:**适用任意数目进程,简单易验证,支持多个临界区,等待耗费CPU时间,不能实现让权等待,可能饥饿和死锁,低效

两个原子操作：

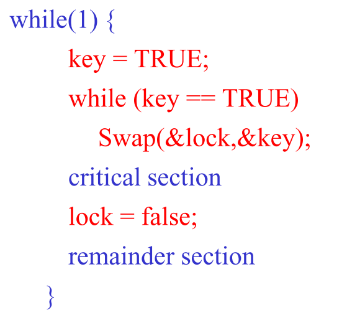




互斥：



Swap互斥



**Spinlock自旋锁**:win/linux内核用来达到多处理器互斥的机制,线程试图获得自旋锁时,CPU上所有其他工作将终止,拥有自旋锁的应用永远不会被抢先.自旋锁应尽快被释放.拥有自旋锁时,执行的指令数将减到最少.

**Semaphores信号量**：记作S,只能进行wait()和signal()两种操作(也被叫做P()和V(),是原子操作).两个相邻wait顺序至关重要,两个相邻signal顺序不重要.**同步：**wait和signal在不同进程出现.**互斥：**wait和signal在同一进程出现.一个同步wait操作与一个互斥wait操作在一起时,同步wait操作在互斥wait操作前.

S.value>0表示S.value个资源可用

S.value=0表示无资源可用且不允许进程再进入临界区.

S.value<0表示有|S.value|个进程在等待队列中或表示进入临界区的进程个数.

wait(S)≡P(S)≡down(S):表示申请一个资源,S--;signal(S)≡V(S)≡up(S):表示释放一个资源,S++.

信号量->互斥

Do{

Wait(mutex);

Signal(mutex);

}while(true);

没有繁忙等待(反复检测一个条件是否为真)的信号量实现:

Block()和wakeup().

Wait:S<0就加到等待队列作block()

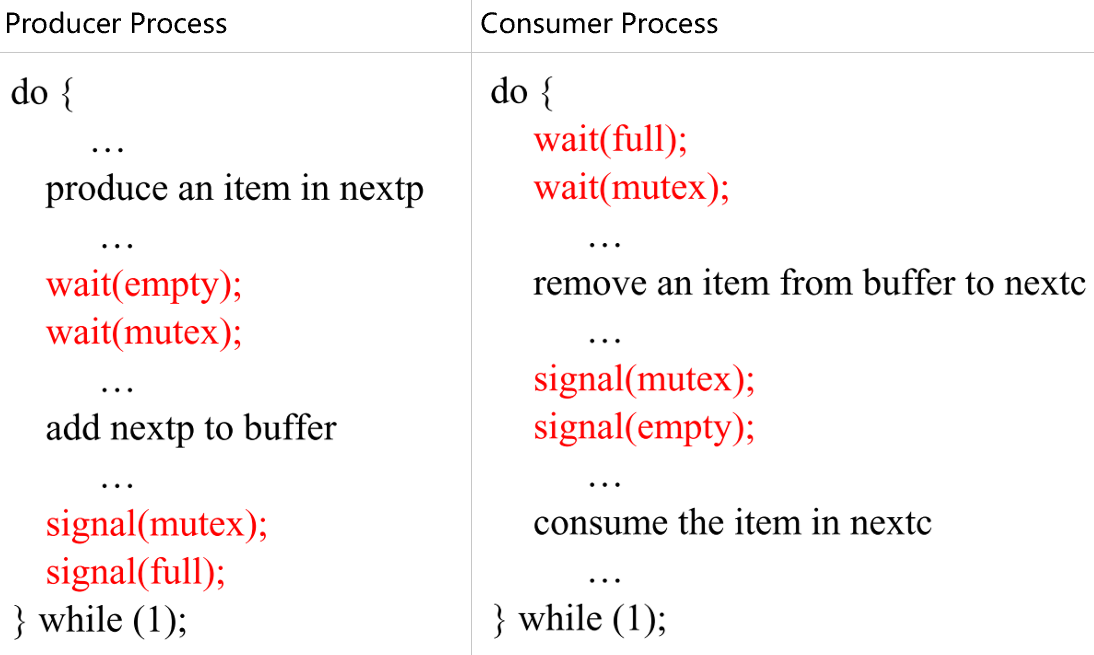
Signal:S<=0就把进程移出等待队列,作wakeup().

**经典同步问题**：

**1.生产者消费者问题(优先缓冲区问题,Bounded-buffer problem)**:它描述一组生产者(P1 ……Pm)向一组消费者(C1……Cq）提供消息.它们共享一个有限缓冲池（bounded buffer pool）,生产者向其中投放消息,消费者从中取得消息.

三个信号量:mutex互斥,初始化为1;

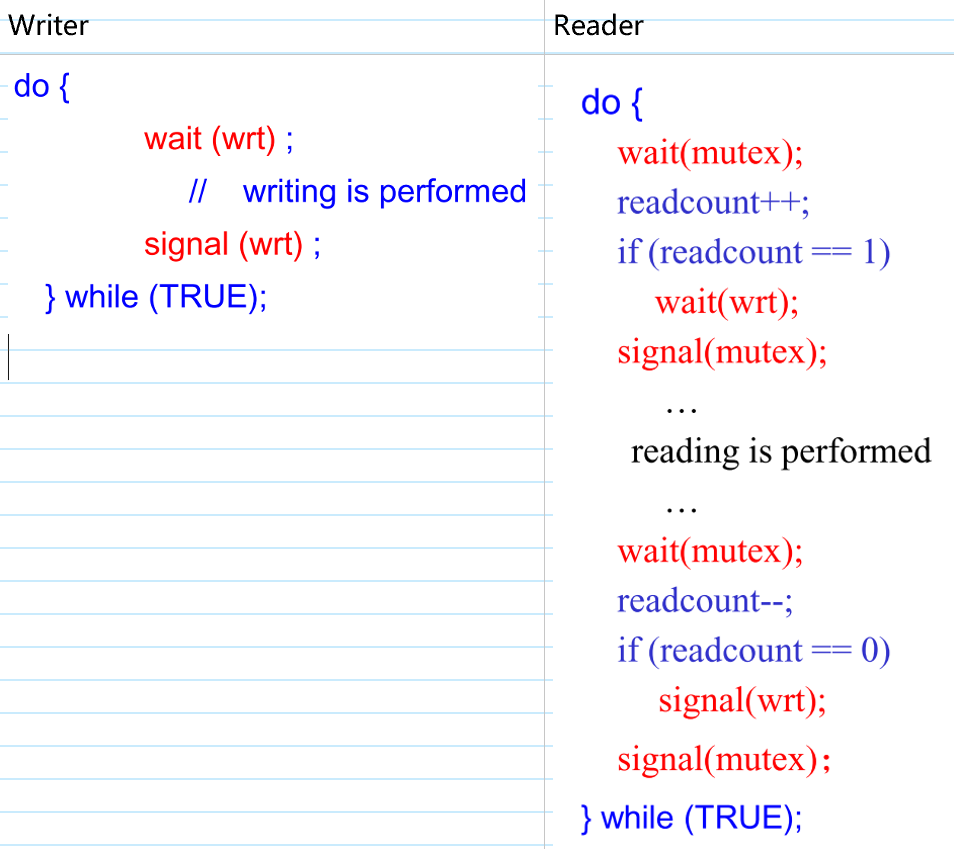
Full满,初始化为0; empty空,初始化为N.



**2.Readers-Writers Problem读者写者问题**：readers只能读,writers可读可写.

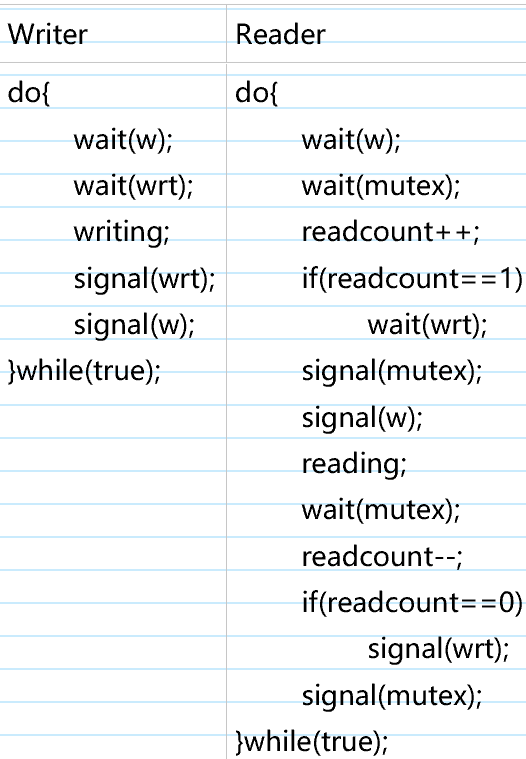
The first readers-writes problem：能并发读不能并发写,写时不能读,读优先.写者可能饥饿.

初始化互斥mutex=1,写wrt=1(读写者互斥),readcount=0(读者计数器)

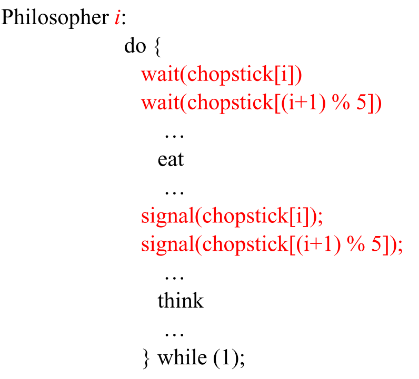


The second readers-writers problem: 能并发读,不能并发写,写时不能读,写优先. 读者可能饥饿.

初始化互斥mutex=1,wrt=1(读写者互斥),w=1(实现写优先), readcount=0(读者计数器)



**3.Dining-Philosophers Problem哲学家就餐问题**：N个哲学家坐在圆桌上,与邻位分享一只筷子.每个哲学家都需要一双筷子去吃,相邻两人不能同时吃,哲学家状态在thinking和eating之间切换.

存在大量死锁！(可能饥饿)

故规定：只允许最多4个人处于thinking态,要么不拿要么一次拿到两只筷子,奇数号人先拿左边的筷子,偶数号人先拿右边的筷子.分成三种状态,thinking,eating和hungry.每个人拿起第一根筷子一段时间后,若拿不到第二根,则放下第一根.

**死锁**

**产生死锁的必要条件**：**1.**互斥 **2.**占有等待 **3.**无抢占 **4.**循环等待,成环

**Resource-Allocation Graph资源分配图**：顶点V和边E, V被分为进程点P和资源点R；E被分为请求边和分配边

如果图没有cycle -> 没有死锁

有cycle：1.每个资源类型只有一个实例,则死锁 2.每个资源类型有好几个实例, 则可能死锁

**死锁定理**：S为死锁状态的充分条件是,当且仅当S状态的资源分配图是不可完全简化的

**死锁预防Deadlock prevention**：

**1.** Mutual Exclusion,共享资源不要求互斥,资源则要求 **2.**Hold and Wait,必须保证当一个进程请求资源时,它不占有其他资源. 要求进程在开始执行前去请求或者被分配资源,或者只有在进程不占有资源时才允许它请求资源;资源利用率低,可能发生饥饿**3.** No preemption,如果持有某些资源的进程请求了不能立刻分配给它的资源,那么它持有的所有资源都会被释放. **4.**Circular Wait,强制所有资源类型的总排序,并要求每个流程以递增的枚举顺序请求资源,资源的有序申请破坏了循环等待条件

**死锁避免Deadlock avoidance**：要求每个进程声明它可能需要的每种资源的最大数量.死锁避免算法动态监测资源分配状态,来保证永远不存在循环等待条件. 资源分配状态由可用资源和已分配资源的数量以及进程的最大需求来定义.

**\*安全状态:**存在一个进程序列,使得系统可以按此分配资源,使进程顺利完成,这被称为安全序列.若不存在,系统处于不安全状态.

**\*安全序列:**

系统处于安全状态->没有死锁

系统不在安全状态->可能死锁

**资源类型具有单个实例**：

**资源分配图Resource allocation graph**：claim edge需求边->request edge请求边->assignment edge分配边->claim edge需求边

只有在需求边变成分配边而不成环时,才允许申请.否则必须等待.

**资源类型具有多个实例**：

**银行家算法The banker’s algorithm:** n=进程数, m=资源类型数

Available[]：available[j]=k表示第j种资源有k个实例

Max[][]：Max[i,j]=k表示进程需要最多k个j种类的资源实例

Allocation[][]：Allocation[i,j]=k表示现在被分配了k个j类的资源实例

Need[][]：Need[i,j]=k表示进程还需要最多k个j类资源实例去完成任务

Need[][]=Max[][]-Allocation[][]

算法：1.初始化Work[]=Available[], Finish[i]=false for i=n-1

2.找到index i满足Finish[i]=false且Need[i]<=Work[i].若不存在,go 4.

3.Work = Work + Allocation[i],Finish[i] = true.(资源释放), go 2.

4.如果所有进程的Finish都为true,说明系统在安全状态.

**问能否给进程分配多少的资源,然后判断系统是否还是安全状态：**

**Resource-request algorithm：**

Request[j]=k,资源请求j类资源k个

1.如果Request[i]<=Need[i],go 2.否则就是错误情况

2.若果Request[i]<=Available[i], go 3.否则必须等待

3.给分配资源：Available-=Request, Allocation+=Request, Need-=Request

4.判断是否是安全状态,用银行家算法

**死锁检测Deadlock detection**：wait for graph等待图,资源分配图的变形,：.检测是否有环,

**死锁检测算法Detection Algorithm**

**1.**初始化Work=available, finish=false如果allocation!=0

**2.**找到一项满足Finish[i]=false and Request[i]<=Work[i]

**3.**Work[i]+=Allocation[i],Finish[i]=true,go 2.

**4.**如果Finish[i]=false,那么系统处于死锁,并且是被死锁的进程

**死锁恢复Deadlock recovery：1.进程终止**,丢弃所有死锁进程或者一次丢弃一个,直到死锁终止 **2.抢占资源**,选择一个牺牲者(最小化代价),回退到安全状态.饥饿问题,会有对象总被选为牺牲者.

**内存管理**

**逻辑地址：**由CPU产生,也被称为虚地址,不能用逻辑地址在内存中读取信息

一对基址寄存器和限长寄存器定义了逻辑地址空间

**物理地址：**内存中存储单元的地址,可直接寻址

**地址绑定的三种情况：1.**Compile time编译时,若内存地址已知,生成绝对码 **2.**Load time编译时不知道,则生成重定位代码relocatable code **3.**Execution time程在运行期间可以从一个内存段移动到另一个内存段,则地址绑定延迟到运行时间.需要硬件支持(如基址寄存器和限长寄存器)

**基址寄存器base register**：包含最小的物理地址的值

**限长寄存器limit register**：包含逻辑地址的范围——每个逻辑地址必须小于限制寄存器（逻辑地址先与其作比较再和relocation register相加）

**MMU,Memory-Management Unit：**将虚地址映射到物理地址的硬件设备. 重定位寄存器relocation register的值被加到每个用户进程产生的地址.用户程序只处理逻辑地址,它永远不会看到真实的物理地址.

**动态载入Dynamic loading：**外部库在进程开始时不会被加载,只有在被需要时才加载

**动态链接Dynamic linking：**外部库可以预加载到（共享）内存中

**交换技术Swapping：**可以将进程暂时从内存中交换到后备存储器backing store,然后将其带回内存继续执行.

Backing store：快速磁盘必须足够大来容纳所有用户的所有内存印象的副本；必须提供对这些内存映像的直接访问

Roll out, roll in(调出,调进)——基于优先级的调度算法

**分区：**1.Multiple-partition allocation将内存划分成若干个连续区域,每个区域只能放一个进程. 2.fixed partition固定分区 3.Dynamic partition动态分区, 在程序装入内存时把可用内存切出一个连续的区域分配给该进程,且分区大小正好适合进程的需要

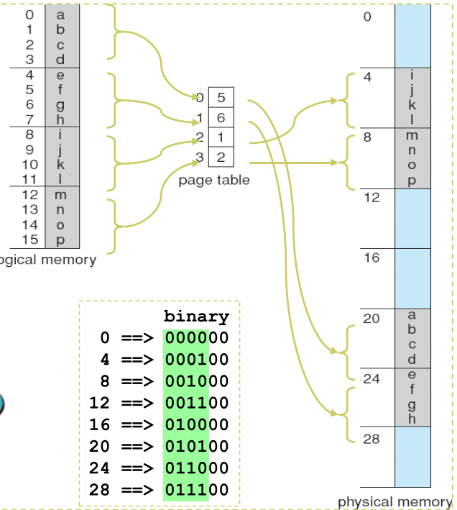
**动态分配问题算法：**有size为n的空间,有一堆free holes. **1. First-fit**, 按顺序查找,分配第一个满足大小的hole **2.Best-fit**, 分配满足要求的holes最小的那一个,必须先搜索整个列表,否则先按大小排好 **3.Worst-fit**,分配最大的hole,必须先搜索整个列表 **4.Next-fit**,总是从上次查找结束的地方开始,类似First-fit

**内存碎片Fragmentation :** 1.internal fragmentation内碎片,单个进程内的碎片,浪费了内存 2.External fragmentation外碎片,进程分区间的碎片,会减少可运行内存的数量.通过压缩或碎片整理(compaction or defragmentation)来减少外碎片

**分页Paging：**

把物理内存分成固定大小的块——frame (帧,物理块,页框）,大小是2的幂,介于512 bytes到8192bytes之间; 把逻辑地址分成固定大小的块,称作页——pages.

逻辑页与物理帧的对应关系



逻辑地址被分成页号page number：对应页表的索引,查询对应的物理内存的基地址和偏移offset：用来在物理内存基地址基础上定位具体地址

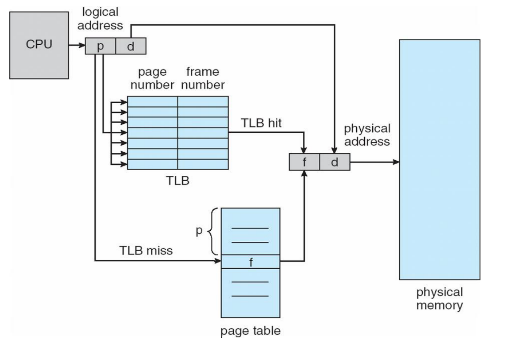
页表的基地址寄存器(Page-table base regiseter, PTBR)指向页表；页表的长度寄存器(Page-table length register, PTLR)保存了页表的大小.

**TLB(translation look-aside buffers)**解决了两次内存访问(页表+data/inst)的问题.有些还有address-space identifiers(ASIDs)地址空间标识符,独一地标识了每个进程,为每个进程提供了地址空间保护.

比如tranlate(p,d) p:虚拟地址 d:物理地址

如果p在TLBs里,可以直接拿对应的frame

如果p不在TLBs里,就要从内存找页表,再去拿对应的frame



设内存访问时间为t, 查表为d,hit率为x

Effective Access time=(t+d)x+(2t+d)(1-x)

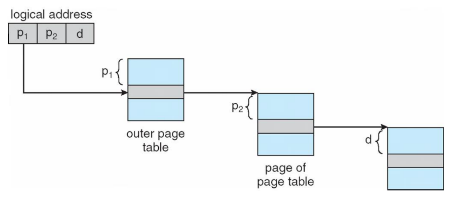
**保护位protection bit**: 与每一帧关联来实现内存保护

**有效位Valid-invalid bit:** 关联页位于进程的逻辑地址空间中,因此是合法页

**Shared page共享页**

**Shared code**:进程间共享的代码,共享代码在所有进程的逻辑地址空间中的地址是一样的

**分级页表Hierarchical Paging:**



32位逻辑地址被分为两个部分

• page number：20bit

• page offset：12bit

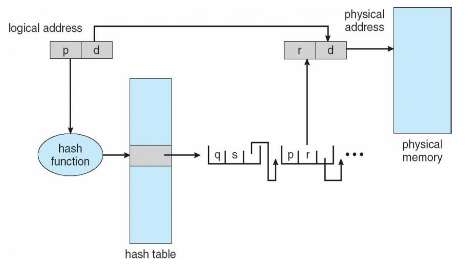
20bit进一步被分为（因为有两级）

• 10-bit page number P1

• 10-bit pageoffset P2

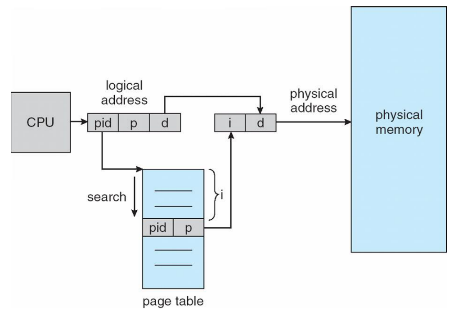
其中是一级页表的page number,是二级页表的page number.

**散列页表Hashed Page Tables:** 将虚拟页码散列到一个页表中,该页表包含一个散列到相同位置的元素链,在这个链中比较虚拟页码以查找匹配项,如果找到匹配项,则提取相应的物理帧



**反向页表Inverted Page Table:** 条目包含存储在实际内存位置的页面的虚拟地址,以及拥有该页面的进程的信息.减少了存储每个页表所需的内存,但是增加了在发生页引用时搜索表所需的时间.使用哈希表将搜索限制为一个或最多几个页表条目.

（由于反向页表按照物理地址排序,而查找的是虚拟地址,因此可能需要查找整个表来寻求匹配.这种查找会花费大量时间）



**分段管理Segmentation**

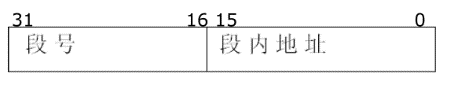
程序是段的集合, 代码共享发生在段级别

Segment table段表

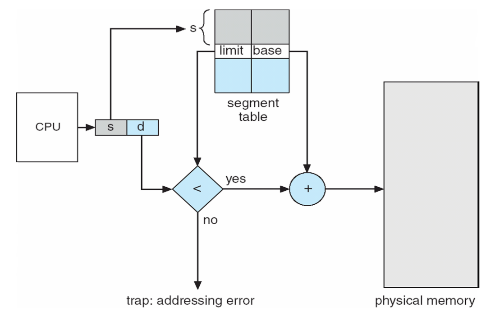
• base：包含起始物理地址

• limit：指定了段的限制长度

段地址：

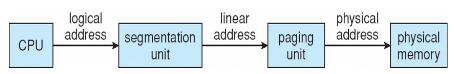


<segment-number, offset>



查找段号->比较offset与limit->计算物理地址base+offset

**地址生成：**CPU生成逻辑地址给分段单元segmentation unit -> 产生线性地址 ->把线性地址给分页单元paging unit -> 产生对应主存里的物理地址



**虚拟内存**

**虚拟内存实现：**1.Demand paging请求页式管理,只有当页被需要时,才把它带入内存 2.Demand segmentation请求段式管理

**Lazy swapper：**除非需要页,否则不要将页交换到内存中

**缺页Page fault：**第一次访问时会出现,有两种可能：1.invalid reference非法地址访问 2.访问的页不在内存中

针对第二种：1.在物理内存建立空白frame,将disk中的页换入该frame 2.重新设置页表 3.重新开始发生错误的指令

Effective memory-access time(**EAT**)：

错误率\*(page fault overhead+swap page out+swap page in+restart overhead)+(1-错误率)\*内存连接时间

**写时拷贝COW(Copy-on-Write):**允许子父进程共享内存中的同一页,如果任何一个进程修改了一个共享页面,那么只有这个页面被复制了

**页面置换Page Replacement:** 当physical memory里没有free frame时,发生Page Replacement页面置换,将内存里一个没有被使用的页面置换出去(没有空页的时候),适用适当的算法——造成尽可能少的page fault产生

**Page replacement algorithm页面置换算法**

**1.FIFO**,first in first out,先进先出. Belady’s Anomaly(异常),可能导致更多的页错误

**2.OPT**,optimal algorithm最佳页面置换算法,选择未来不再使用的页或离当前最远位置上出现的页来置换

**3.LRU**,Least Recently Used Algorithm最近最少使用算法,选择内存中最久没有引用的页置换.需要记录页使用时间,开销大.如何记录？Counter复制使用的时刻或Stack引用后页放到最上面

**4.LRU Approximation Algorithms,**近似LRU算法. **1.**Reference bit引用位,初始为0,当被引用,置1.替换那些为0的页,顺序不一定. **2.** Additional-Reference-Bits Algorithm每一页有一个8位字节,被访问时最高字节置1,定期整体右移一位,于是寄存器数值最小的是最久未使用页面 **3.**Second-Chance(clock) Algorithm, 按时间顺序替换页前,检查页的reference bit是否为1.如果为1,将reference bit置0,插入队列尾,像新的对象一样.如果reference bit为0,替换该页 **4.** Enhanced Second-Chance Algorithm,使用引用位和修改位(reference bit, modified bit)：(0,0), (0,1), (1,0), (1,1)被替换优先级依次下降, MAC使用.

**5.Counting-Base Page Replacement**：对每个page的引用次数有记录. **LFU(Least Frequently Used)最不经常使用替换, MFU (Most Frequently Used)最经常使用替换**

**6.Page Buffering Algorithm页面缓冲算法**,用FIFO算法选择被置换页,被置换页未被修改：将其放入空闲页面链表末尾；被置换页已被修改：将其放入已修改页面链表.调入新页面时,将新页面内容读入到空闲页面页表的第一项,然后将原来的第一项删除.两个链表存储的页面在内存中停留一段时间,如果这些页面被再次访问,这些页面还在内存中.当已修改页面达到一定数目后,将它们一起调出到外存,然后将它们归入空闲页面链表.Win/Linux基于该算法.

**帧分配Allocation of frames**：1.固定分配,平均或加权 2.优先级分配,从自己的frames或从低优先级的frames里选一个替换 3.Global replacement全局置换,可从所有的frames里选择 4.Local replacement局部置换,只能从自己的frame里选

1.固定分配局部置换策略

2.可变分配全局置换策略

3.可变分配局部置换策略

**Thrashing颠簸、抖动:**进程没有足够的页,page fault rate很高,忙于交换页

导致低CPU使用率, OS认为需要提高degree of multiprogramming,将别的进程引入,恶化状况

**working set(WS)工作集**：在某段间隔Δ里,进程实际要访问的页面的集合

Δ≡working-set window工作集窗口≡页面访问的一个fixed number, 把在某段时间间隔Δ,时间t之前的工作集记为w(t,Δ)

（working set size of Process）工作集大小=total number of pages referenced in the most recent Δ: 如果Δ太小,就不能表现进程的局部特征;如果Δ太大,工作集就会趋向于进程执行需要的所有页面的集合

D==所有请求的frames

m=所有available的frames

如果D>m ⇒ Thrashing

策略：挂起一个优先级低的进程

**Buddy System**:从物理上连续的页组成的固定大小段分配内存,内存被分成2的幂次的大小,当真正需要的内存小于拥有的内存块时,对半切割内存块,直到最合适.

**Slab Allocator:** Slab是物理上一个或多个连续的页,Cache由一个或多个Slab构成,一个cache对应一个独特的内核数据结构,每个cache都被对象充满——对象是数据结构的实例化,cache创建时,被free objects填满.结构存储时,object被标记为used.如果一个slab充满了used objects,下一个object会被分配到一个空的slab.如果没有空的slab,那就创建新的slab.这样做的好处是,没有碎片,内存请求被快速满足

**预调页Prepaging**：为了降低程序开始时大量的page fault,在开始前就准备好部分或者全部的程序会用到的页.但如果有prepared page没有被用到,就会造成memory的浪费.

**TLB范围reach**：TLB可访问的内存量=TLB Size×Page Size.理想情况下.工作集应该完全被存在TLB中.提高page size：可能会增加碎片.提供不同的page size：比较好.造成的碎片较少.

**I/O联锁interlock**：用于从设备中复制文件的页面必须被锁定.否则将被页面替换算法选中进行清除

**文件系统**

**文件:**a file is a sequence of bytes stored on some device. 属性：文件名 标识符 种类 位置 大小 保护(控制读写权限) 时期日期

**打开文件**：从外部存储读取文件控制信息到内存中.文件Close的实质是把内容从内存移出到disk

**文件内部结构**：流文件,记录文件结构,复杂结构(格式化文档,可重定位载入档).可以通过插入适当的控制字符,用第一种方法模拟后两种结构

**访问方法：**1.顺序存取,只能按顺序来2.直接存取,可选择想要的位置 3.索引顺序,根据索引查找文件然后读取

包含文件系统的实体被称为**卷(volume)**

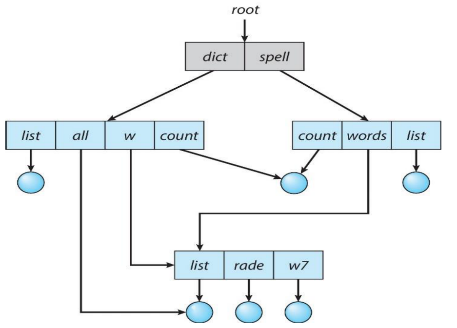
**目录结构**

**1.单级目录Single-Level Directory**,一个面向所有用户的单级目录,命名和分组问题

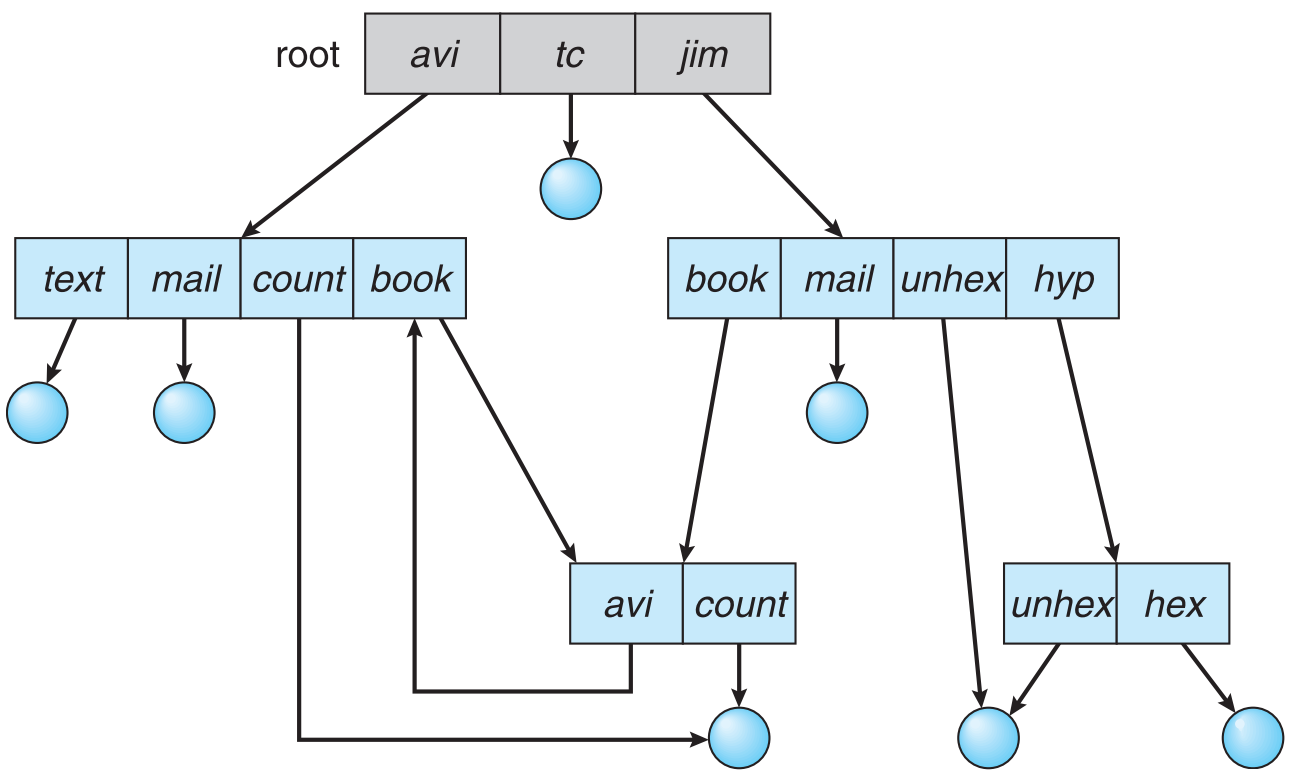
**2.Two-level Directory二级目录**,按用户分组,不同用户的文件可以有相同的文件名,没有分组功能

**3.** **Tree-Structured Directories树型结构**,即我们现在在用的,绝对/相对路径

**4.Acyclic-Graph Directories 无环图结构目录**,有共享子目录和文件, 删除操作可能只造成指针的置空,没有真正删除文件, Entry-hold-count solution表项保留计数,计数为0时删除文件. 硬链接

****

**5.** **General Graph Directory普通图结构目录**:如何保证无环？只允许链接到文件,而不能是子目录;Garbage collection机制;每次添加一个新的链接都使用环检测算法来检验是不是好的



**File system mounting文件系统挂载:**文件系统在能被access前必须被挂载;一个未被挂载的文件系统被挂载在一个挂载点(mount point)上

**File sharing文件共享:**Network File System（NFS）是一个常见的分布式文件共享方法,user IDs and Group IDs

**Consistency Semantics一致性语义(可以理解为同步）:1.**UNIX file system(UFS)**,**对一个打开文件的写入操作,将立即对其他用户可见;共享文件指针,允许多个用户同时读写2.Andrew File System(AFS),写入操作只对文件关闭后开始的会话可见

**Protection保护：**RWX权限 可用数字0-7表示 7权限最高,三类,owner group public

Chmod 761 game, game是文件名

**FCB(file control block)文件控制块**,在open文件时被创建你

**On-Disk File system包含信息：**如何启动所存储的操作系统;总的块数;空闲块的数目和位置;目录结构以及各个具体文件等. Ext2

**In-Memory File Systen包含信息:**An in-memory partition table分区表,An in-memory directory structure目录结构,The system-wide open-file table系统打开文件表,The per-process open-file table进程打开文件表.e.g. Linux vfs(对不同文件系统提供相同系统调用接口API)

**Allocation Methods文件物理结构**：

**1.Contiguous Allocation连续分配,**不适合存储大小动态增长的文件

**2.** **Linked Allocation链接分配,**链表形式,随机访问不行. FAT12/16/32数字代表地址位数, File-Allocation Table.FAT32目录项结构,一个目录项32个字节,一个表项存放短文件名和其他属性,长文件名的目录项由几个32B表项组成.NTFS文件系统,每个分区都有一个Master File Table(MFT),每个文件占用一个记录项,每个文件或目录的信息都包含在MFT里

**3.** **Indexed Allocation索引分配**,把所有的指针放在一个索引块里,二级索引Two-level index

**大容量存储系统**

磁盘的0扇区是最外面的第一个磁道的第一个扇区。逻辑块时最小传出单位512B

**磁盘Access time**:Seek time寻道时间, rotational latency旋转延迟, Transfer time传输时间

**磁盘调度算法**

**1.FCFS**,先来先服务**2.SSTF**,Shortest seek time first最短寻道时间优先 **3.SCAN (elevator algorithm)**扫描,向一端前进,然后调转方向向另一端前进**4.C-SCAN循环扫描**,向一端前进,然后不扫描快速到另一端,重新开始扫描**5.C-LOOK**,只会在一个方向上执行到最后一个请求为止,不到达末尾,然后反转方向达到另一端的第一个请求

SSTF,common used

SCAN, C-SCAN,对荷载大的系统效果好

**Boot：**一般bootstrap存在ROM里。系统启动顺序：ROM中的代码(simple bootstrap)boot block里的代码(full bootstrap)也就是boot loader如Grub LILO然后是整个OS内核

**RAID冗余廉价磁盘阵列：**RAID 0：最快最有效率,但不支持容错;RAID 1：性能高有容错,在少于2个disk时的唯一选择。n/2块盘作镜像;RAID 3：数据加强,加速单用户对连续长记录时的数据传输。一块盘储存校验码;RAID 5：多用户,对数据写入性能要求不高时的唯一选择。校验码在所有盘上均匀分布;RAID 6：5的基础上,又增加了异种校验码;RAID 10：高性能,可靠

**三级存储Tertiary storage device：**Low cost is the defining characteristic of tertiary storage. Generally, tertiary storage is built using removable media Common examples of removable media are floppy disks and CD-ROMs; other types are available

**Swap space三级存储Tertiary storage device:** 虚存使用硬盘空间作为主存。两种形式：普通文件系统：win都是pagefile.sys 独立硬盘分区linux solaris都是swap分区。还有一种方法：创建在raw的磁盘分区上，这种速度最快。

**I/O系统**

**I/O方式:**Polling轮询,不常用;Interrupt中断

**DMA:**Direct Memory Access,绕过CPU在I/O和内存间传输数据

**I/O系统调用, Device-driver设备驱动**：实现统一的I/O接口

**Blocking阻塞I/O**:进程挂起直到I/O完成

**Nonblocking非阻塞I/O**:I/O调用立刻返回，通过多线程完成

**OAsynchronous异步**:进程与I/O同时运行,difficult to use

非阻塞与异步的区别：非阻塞read调用会马上返回，其读取的数据可以少于要求的或为0; 异步read要求传输完整执行

**Buffering缓冲:**保存两设备之间或设备与应用之间传输的数据,解决了设备速度不匹配的问题（CPU与I/O速度不匹配）,解决设备传输块大小不匹配问题