内存管理方式

一.分页存储

用户程序的地址空间被划分成若干固定大小的区域称为页，相应内存空间分成若干物理块，页和块的大小相等，可将程序的任一页放入内存的任一块中，实现了离散分配，运行时一页一页读取

等分内存：将内存空间划分为等长的若干物理块，每个物理块的大小一般取为2的整数幂，所有物理块从0开始编号，称为物理页号，相邻的页面在内存中不一定相邻，即分配给程序的内存块之间不一定连续

逻辑地址：系统将程序的逻辑空间按照同样大小也划分成若干页面，逻辑页面从0开始依次编号，称为逻辑页号或相对页号，每个页面内从0开始编址称为页内地址。程序中逻辑地址有两部分：页号p和页内位移量w

页号x位表示每个作业最多2的x次方页了，页内位移量y位表示页的大小为2的y次方

逻辑地址到物理地址的变换：系统为每个进程建立一张页表，用于记录逻辑页面与内存物理页面的对应关系（页号到物理块号的地址映射）

页号：逻辑地址/页面大小　页内偏移：逻辑地址%页面大小

物理地址=物理快号*页面大小+页内偏移

页式管理的优点：

- 没有外碎片
- 一个程序不必连续存放
- 便于改变程序占用空间大小

页式管理的缺点：

- 无论数据量大小都只能按照页面大小分配，容易产生内部碎片（页面可能填不满造成浪费）
- 不能体现程序逻辑
- 页长与程序的逻辑大小不相关
- 不利于编程时的独立性，并给换入换出处理、存储保护和存储共享等操作造成麻烦

二.分段存储

将用户地址空间划分成若干个不相等的段，每段可以定义一组相对完整的逻辑信息，存储分配时以段为单位，段与段在内存中也可以不相邻接实现了离散分配

分段地址结构：段的长度由相应的逻辑信息组的长度决定，因而各段长度不等

逻辑地址：在段式虚拟存储系统中，逻辑地址由段号和段内地址构成，逻辑地址到物理地址的变换通过段表来实现

地址映射：针对每一个虚拟地址，首先以段号S为索引访问段表的第S个表项，将虚拟地址的段内地址与该表项的段长字段比较；若段内地址较大则说明地址越界，将产生地址越界中断；否则，物理地址就为该表项的段起址+段内地址

分段存储的优点：

- 段的逻辑独立性使其易于编译、管理、修改和保护，也便于多道程序共享。
- 段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。
- 方便编程，分段共享，分段保护，动态链接，动态增长

分段存储的缺点：

- 主存空间分配比较麻烦。
- 容易在段间留下许多碎片（外部碎片），造成存储空间利用率降低。
- 由于段长不一定是2的整数次幂，因而不能简单地像分页方式那样用虚拟地址和实存地址的最低若干二进制位作为段内地址，并与段号进行直接拼接，必须用加法操作通过段起址与段内地址的求和运算得到物理地址。因此，段式存储管理比页式存储管理方式需要更多的硬件支持。

三.分页与分段的主要区别

- 页是信息的物理单位，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率；段则是信息的逻辑单位，它含有一组其意义相对完整的信息，分段的目的是为了能更好地满足用户的需要。

- 页的大小固定且由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面；而段的长度却不固定，决定于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。
- 分页的作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个记忆符，即可表示一个地址；而分段的作业地址空间则是二维的，程序员在标识一个地址是，即需给出段名，又需给出段内地址。
- 分页信息很难保护和共享、分段存储按逻辑存储所以容易实现对段的保存和共享。

四.段页式存储

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。它首先将程序按其逻辑结构划分为若干个大小不等的逻辑段，然后再将每个逻辑段划分为若干个大小相等的逻辑页。主存空间也划分为若干个同样大小的物理页。辅存和主存之间的信息调度以页为基本传送单位，每个程序段对应一个段表，每页对应一个页表。

段页式地址结构:程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块段页式系统中，作业的地址结构包含三部分的内容：段号，页号，页内位移量CPU访问时，段表指示每段对应的页表地址，每一段的页表确定页所在的主存空间的位置，最后与页表内地址拼接，确定CPU要访问单元的物理地址。段页存储管理方式综合了段式管理和页式管理的优点，但需要经过两级查表才能完成地址转换，消耗时间多。

地址映射：

- 进行地址变换时，首先利用段号S，将它与段表长TL进行比较。若S<TL，表示未越界。
- 于是利用段表始址和段号来求出该段所对应的段表项在段表中的位置，从中得到该段的页表始址

- 利用逻辑地址中的段内页号P来获得对应页的页表项位置，从中读出该页所在的物理块号b
- 再利用块号b和页内地址来构成物理地址。

段页式存储的优点：

- 它提供了大量的虚拟存储空间。
- 能有效地利用主存，为组织多道程序运行提供了方便。

缺点：

- 增加了硬件成本、系统的复杂性和管理上的开消。
- 存在着系统发生抖动的危险。
- 存在着内碎片。
- 还有各种表格要占用主存空间。

死锁

一.死锁条件

- 互斥条件:一个资源每次只能被一个进程使用
- 不可剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺
- 请求与保持条件:一个进程因请求资源而阻塞时，对已获得的资源保持不放
- 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系.

二.死锁预防

破坏互斥条件:允许某些进程(线程)同时访问某些资源，但有的资源不允许同时被访问如打印机等。

破坏不可抢占条件:即允许进程强行从占有者那里夺取某些资源。这种预防方法实现起来困难，会降低系统性能。

破坏占有且申请条件:可以实行预先分配策略，即进程在运行前一次性地向系统申请它所需要的全部资源。如果当前进程所需的全部资源得不到满足，则不分配任何资源。只有当系统能够满足当前的全部资源得到满足时，才一次性将所有申

请的资源全部分配给该进程。由于运行的进程已占有了它所需的全部资源，所以不会发生占有资源又重新申请资源的现象，因此不会发生死锁。但是有以下缺点：

- 在许多情况下，一个进程在执行之前不可能知道它所需的全部资源。这是由于进程在执行时是动态的，不可预测的。
- 资源利用率低。无论所分配资源何时用到，一个进程只有在占有所需的全部资源后才能执行。即使有些资源最后才被该进程用到一次，但该进程在生存期间一直占有它们，造成长期占有。
- 降低了进程的并发性。因为资源有限，又加上存在浪费，能分配到所需全部资源的进程个数必然少了。

破坏循环等待条件。实行资源有序分配策略。采用这种策略即把资源事先分类编号，按号分配。所有进程对资源的请求必须严格按资源需要递增的顺序提出。进程占用好小资源，才能申请大号资源，就不会产生环路。这种策略与前面的策略相比，资源的利用率和系统吞吐量都有很大提高，但是也存在以下缺点：限制了进程对资源的请求，同时系统给所有资源合理编号也是件困难事，并增加了系统开销。

三.死锁的避免

1. 银行家算法：该算法需要检查申请者对资源的最大需求量，如果系统现存的各类资源可以满足申请者的请求，就满足申请者的请求。这样申请者就可很快完成其计算，然后释放它占用的资源，从而保证了系统中的所有进程都能完成，所以可避免死锁的发生。

四.死锁的解除

- 资源剥夺法：挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源，而处于资源匮乏的状态。
- 撤销进程法：强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。

- 进程回退法：让一（多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。

进程间通信方式

- **无名管道(pipe)**：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。管道是单向的、先进先出的、无结构的、固定大小的字节流，它把一个进程的标准输出和另一个进程的标准输入连接在一起。写进程在管道的尾端写入数据，读进程在管道的首端读出数据。数据读出后将从管道中移走，其它读进程都不能再读到这些数据。
- **命名管道 (named pipe)**：命名管道也是半双工的通信方式，它克服了管道没有名字的限制，并且它允许无亲缘关系进程间的通信。命令管道在文件系统中有对应的文件名，命名管道通过命令mkfifo或系统调用mkfifo来创建。
- **信号量(semophore)**：信号量是一个计数器，可以用来控制多个进程对共享资源的访问实现进程间的互斥与同步。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段，信号量基于操作系统的PV操作，程序对信号量的操作都是原子操作
- **消息队列(message queue)**：消息队列是由消息的链表结构实现，存放在内核中并由消息队列标识符标识。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。
- **信号 (sinal)**：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。除了用于进程通信外，进程还可以发送信号给进程本身。
- **共享内存(shared memory)**：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。

共享内存是最快的IPC方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量配合使用，来实现进程间的同步和通信。

- **套接字(socket)：** 也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

进程调度算法

1.进程调度的原因

在操作系统中，由于进程总数多于处理机，它们必然竞争处理机。为了充分利用计算机系统中的CPU资源，让计算机系统能够多快好省地完成我们让它做的各种任务，所以需要进行进程调度。

2.进程调度的定义

进程调度（也称CPU调度）是指按照某种调度算法（或原则）从就绪队列中选取进程分配CPU，主要是协调对CPU的争夺使用。

通常有以下两种进程调度方式：

- 非剥夺调度方式，又称非抢占方式：是指当一个进程正在处理机上执行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在执行的进程继续执行，直到该进程完成或发生某种事件而进入阻塞状态时，才把处理机分配给更为重要或紧迫的进程。在非剥夺调度方式下，一旦把CPU分配给一个进程，那么该进程就会保持CPU直到终止或转换到等待状态。这种方式的优点是实现简单、系统开销小，适用于大多数的批处理系统，但它不能用于分时系统和大多数的实时系统。
- 剥夺调度方式，又称抢占方式：是指当一个进程正在处理机上执行时，若有某个更为重要或紧迫的进程需要使用处理机，则立即暂停正在执行的进程，将处理机分配给这个更为重要或紧迫的进程。

3.进程调度的常见算法

1）先来先服务调度算法（FCFS，First Come First Server)

处于就绪态的进程按先后顺序链入到就绪队列中，而FCFS调度算法按就绪进程进入就绪队列的先后次序选择当前最先进入就绪队列的进程来执行，直到此进程

阻塞或结束，才进行下一次的进程选择调度。FCFS调度算法采用的是不可抢占的调度方式，一旦一个进程占有处理机，就一直运行下去，直到该进程完成其工作，或因等待某一事件而不能继续执行时，才释放处理机。

FCFS调度算法的特点是算法简单，但效率低；对长作业比较有利，但对短作业不利（相对SJF和高响应比）；有利于CPU繁忙型作业，而不利于I/O繁忙型作业。

2）短作业优先调度算法（SJF，Short Job First)

短作业（进程）优先调度算法是指对短作业（进程）优先调度的算法。短作业优先(SJF)调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先(SPF)调度算法，则是从就绪队列中选择一个估计运行时间最短的进程，将处理机分配给它，使之立即执行，直到完成或发生某事件而阻塞时，才释放处理机。

SJF又分为两种：

（1）SRTF抢占式：又称最短剩余优先，当新进来的进程的CPU区间比当前执行的进程所剩的CPU区间短，则抢占。

（2）非抢占：称为下一个最短优先，即为在就绪队列中选择最短CPU区间的进程放在队头。

SJF调度算法的特点是吞吐率高，平均等待时间、平均周转时间最少；但算法对长作业十分不利，也完全未考虑作业的紧迫程度。

3）时间片轮转法（RR，Round Robin)

时间片轮转调度算法主要适用于分时系统。在这种算法中，系统将所有就绪进程按到达时间的先后次序排成一个队列，进程调度程序总是选择就绪队列中第一个进程执行，即先来先服务的原则，但仅能运行一个时间片，如100ms。在使用完一个时间片后，即使进程并未完成其运行，它也必须释放出（被剥夺）处理机给下一个就绪的进程，而被剥夺的进程返回到就绪队列的末尾重新排队，等候再次运行。

在时间片轮转调度算法中，时间片的大小对系统性能的影响很大。如果时间片足够大，以至于所有进程都能在一个时间片内执行完毕，则时间片轮转调度算法就退化为先来先服务调度算法。如果时间片很小，那么处理机将在进程间过于频繁

切换，使处理机的开销增大，而真正用于运行用户进程的时间将减少。因此时间片的大小应选择适当。

时间片的长短通常由以下因素确定：系统的响应时间、就绪队列中的进程数目和系统的处理能力。

时间片q = 系统对相应时间的要求RT / 最大进程数N

（经验表明，时间片的取值，应该使得80%的进程在时间片内完成所需的一次CPU运行活动。）

4）多级反馈队列调度算法（MLFQ，Multi-Level Feedback Queue)

设置多个就绪队列，并为各个队列赋予不同的优先级。第一个队列的优先级最高，第二队次之，其余队列优先级依次降低。仅当第1～i-1个队列均为空时，操作系统调度器才会调度第i个队列中的进程运行。赋予各个队列中进程执行时间片的大小也各不相同。在优先级越高的队列中，每个进程的执行时间片就越小或越大（Linux-2.4内核就是采用这种方式）。

当一个就绪进程需要进入就绪队列时，操作系统首先将它放入第一队列的末尾，按FCFS的原则排队等待调度。若轮到该进程执行且在一个时间片结束时尚未完成，则操作系统调度器便将该进程转入第二队列的末尾，再同样按先来先服务原则等待调度执行。如此下去，当一个长进程从第一队列降到最后一个队列后，在最后一个队列中，可使用FCFS或RR调度算法来运行处于此队列中的进程。

如果处理机正在第i（i>1）队列中为某进程服务时，又有新进程进入第k（k<i）的队列，则新进程将抢占正在运行进程的处理机，即由调度程序把正在执行进程放回第i队列末尾，重新将处理机分配给处于第k队列的新进程。

5）高响应比优先调度算法（HRRF，Highest Response Ratio First)

高响应比优先调度算法主要用于作业调度，该算法是对FCFS调度算法和SJF调度算法的一种综合平衡，同时考虑每个作业的等待时间和估计的运行时间。在每次进行作业调度时，先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入运行。

响应比 = （等待时间+要求服务时间）/ 要求服务时间 = 响应时间 / 执行时间

6）最高优先级优先调度算法（PR，Priority First)

在作业调度中，最高优先级调度算法每次从后备作业队列中选择优先级最高的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。

在进程调度中，优先级调度算法每次从就绪队列中选择优先级最高的进程，将处理机分配给它，使之投入运行。
根据新的更高优先级进程能否抢占正在执行的进程，可将该调度算法分为：

- 非剥夺式优先级调度算法：当某一个进程正在处理机上运行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在运行的进程继续运行，直到由于其自身的原因而主动让出处理机时（任务完成或等待事件），才把处理机分配给更为重要或紧迫的进程。
- 剥夺式优先级调度算法：当一个进程正在处理机上运行时，若有某个更为重要或紧迫的进程进入就绪队列，则立即暂停正在运行的进程，将处理机分配给更重要或紧迫的进程。

而根据进程创建后其优先级是否可以改变，可以将进程优先级分为以下两种：
静态优先级：优先级是在创建进程时确定的，且在进程的整个运行期间保持不变。确定静态优先级的主要依据有进程类型、进程对资源的要求、用户要求。
动态优先级：在进程运行过程中，根据进程情况的变化动态调整优先级。动态调整优先级的主要依据为进程占有CPU时间的长短、就绪进程等待CPU时间的长短。