

操作系统总结

第一章 概论

- 操作系统：计算机操作系统是指控制和管理计算机的软、硬件资源，合理组织计算机的工作流程，方便用户使用的程序集合。
- 目标：方便性、有效性、可扩充性、开放性。
- 发展历程（大概了解）：
 - 人工操作：用户独占全机、CPU等待人工操作。
 - 单道批处理系统：内存中始终只保持一道作业，系统资源得不到充分的利用。
 - 多道批处理系统：用户提交的作业存放在外存上排成一个队列，由作业调度程序按一定的算法，从后备队列中选择若干个作业调入内存，使它们共享CPU和系统中的各种资源。
 - 优点：资源利用率高、系统吞吐量大。
 - 缺点：平均周转时间长、无交互能力。
 - 分时系统：多个用户分享使用同一台计算机，多个程序分时共享硬件和软件资源，通常按时间片来分配各个程序在CPU上执行的轮换时间。
 - 特征：
 - 多路性：系统允许将多台终端同时连接到一台主机上，并按分时原则为每个用户服务。
 - 独立性：每个用户在各自的终端上进行操作，彼此之间互不影响。
 - 及时性：用户的请求能在很短时间内获得响应。
 - 交互性：用户可通过终端与系统进行广泛的人机对话。
 - 实时系统：系统能及时响应外部事件的请求，在规定的时间内完成对该事件的处理，并控制所有实时任务协调一致地运行。
- 功能：
 - 作为用户与计算机硬件系统之间的接口：用户在OS帮助下能够方便、快捷、可靠地操纵计算机硬件和运行自己的程序。
 - 作为计算机系统资源的管理者：OS可以对计算机四类资源：处理机、存储器、I/O设备以及文件进行有效管理。
 - 实现了对计算机资源的抽象：不仅增强了系统的功能，还隐藏了对硬件操作的具体细节，实现了对计算机硬件操作的多个层次的抽象模型。
- 特征（并发和共享是最基本的特征）：
 - 并发性：在操作系统中同时存在许多活动，多个事件会在同一时间段内发生。
 - 并行：两个或多个事件在同一时刻发生（区别）
 - 共享性：系统中的资源可供内存中多个并发执行的进程共同使用。
 - 互斥共享：一段时间只允许一个进程访问（临界资源）
 - 同时访问：一段时间内允许多个进程同时访问。
 - 虚拟性：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。
 - 举例：虚拟处理机、虚拟存储器、虚拟设备技术。
 - 异步性：指进程的执行顺序和执行时间的不确定性。
- 分类：批处理操作系统、分时系统、实时操作系统、网络操作系统、分布式操作系统、多处理机操作系统、嵌入式操作系统

第二章 进程

1.概念

进程：是具有独立功能的程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位。

- 进程特征：
 - 动态性：进程是程序的一次执行，具有一定的生命期，是动态地产生、变化和消亡的。
 - 并发性：进程之间的动作在时间上可以重叠。
 - 独立性：进程是系统调度和资源分配的独立单位，它具有相对独立的功能，拥有自己独立的进程控制块PCB。
 - 异步性：各个并发进程按照各自独立的、不可预知的速度向前推进。
 - 交互性：并发进程之间具有直接或间接的关系，在运行过程中需要进行必要的交互，以完成特定的任务。
- 程序与进程的区别（简答题）：
 - 程序是静态的，进程是动态的
 - 进程与程序的组成不同，进程 = 程序 + 数据 + PCB
 - 进程的存在是暂时的，程序的存在是永久的
 - 一个程序可以对应多个进程，一个进程可以包含多个程序

2.进程的表示和状态转换

- 进程控制块PCB：系统为了管理进程设置的一个专门的数据结构，用来记录进程的外部特征，描述进程的变化过程。
- PCB是系统感知进程存在的唯一标志，进程与PCB是一一对应的。
- PCB主要包含以下信息：
 - 进程标识符：用于唯一表示一个进程，包含外部标识符和内部标识符。
 - 处理机状态：由处理机的各种寄存器中的内容组成，包括通用寄存器R、指令寄存器PC、程序状态字PSW、用户栈指针SP等。
 - 进程调度信息：有关进程的状态及有关进程调度的信息，包括：进程状态、进程优先级、进程调度所需的其它信息。
 - 进程控制信息：用于进程控制所必须的信息，包括：程序和数据的地址、进程同步和通信机制、资源清单、链接指针。
- PCB组织方式（了解）：
 - 线性方式：将所有PCB组织在一张线性表中。
 - 链接方式：通过PCB中的链接字链接成一个队列。
 - 索引方式：根据所有进程状态的不同建立几张索引表。
- 最基本的进程状态有三种：
 - 运行状态：进程占有CPU，并在CPU上运行
 - 就绪状态：一个进程已经具备运行条件，但由于无CPU暂时不能运行的状态（当调度给其CPU时，立即可以运行）
 - 阻塞（等待）状态：指进程因等待某种事件的发生而暂时不能运行的状态（即使CPU空闲，该进程也不可运行）
- 增加的状态：
 - 创建状态：进程刚创建，但还不能运行
 - 结束状态：进程已结束运行，回收除PCB之外的其他资源，并让其他进程从PCB中收集有关信息
 - 挂起状态：一些低优先级进程可能等待较长时间而被对换至外存，为运行进程提供足够内存
 - 阻塞挂起：进程在外存并等待某事件的出现。
 - 就绪挂起：进程在外存，但只要进入内存即可运行。

无挂起的状态图：

七状态图：

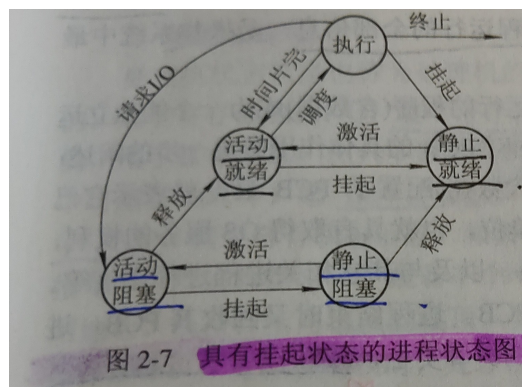


图 2-7 具有挂起状态的进程状态图

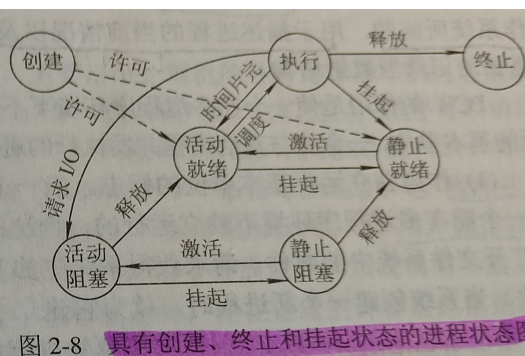


图 2-8 具有创建、终止和挂起状态的进程状态图

- 创建→活动就绪：在系统的性能和内存容量允许的情况下，进程的状态会转换成就绪状态。
- 创建→静止就绪：若系统当前资源状况和性能要求不允许的情况下，相应程序状态将转化为静止就绪被安置在外存，不参与调度。

3.进程控制

OS内核：将一些与硬件紧密相关的模块、各种常用设备的驱动程序以及运行频率较高的模块都安排在紧靠硬件的软件层次中，将它们常驻内存。

- OS内核一般都包含两大功能：
 - 支撑功能：
 - 中断处理
 - 时钟管理：如时间片轮转调度
 - 原语操作：原语就是由若干条指令组成，用于完成一定功能的一个过程。
 - 资源管理功能：
 - 进程管理
 - 存储器管理
 - 设备管理

(1) 进程创建

- 创建事件：
 - 用户登录
 - 作业调度
 - 提供服务
 - 应用请求
- 创建步骤：
 - 申请空白PCB，为新进程申请获得唯一的数字标识符
 - 为新进程分配其运行所需的资源
 - 初始化进程控制块PCB：初始化标识信息、初始化处理机状态信息、初始化处理机控制信息
 - 如果进程就绪队列能够接纳新进程，则将其插入就绪队列

(2) 进程终止

- 终止事件：
 - 正常结束
 - 异常结束
 - 外界干预
- 终止过程：

- 找到相应进程的PCB
- 若进程正处于执行状态，则立即停止，设置重新调度标志
- 撤消属于该进程的所有“子孙”进程
- 释放被撤消进程的所有资源
- 释放进程的PCB
- 若调度标志为真，则进行重新调度

(3) 程序阻塞与唤醒（等待）

- 引起阻塞与唤醒事件：
 - 向系统请求共享资源失败
 - 等待某种操作的完成
 - 新数据尚未到达
 - 等待新任务到达
- 阻塞过程：
 - 找到相应进程PCB
 - **若为执行状态**：保护其现场，将其状态改变为等待状态，停止运行，并把该PCB插入到相应的等待队列中去
 - **若为就绪状态**：将其状态修改为等待状态，把它移出就绪队列，并插入到等待队列中去
- 唤醒过程：
 - 在等待队列中找到相应进程的PCB，将其从等待队列中移出
 - 置其状态为就绪状态，然后把该PCB插入就绪队列中
 - 等待调度程序调度

4.进程同步/互斥

进程同步机制：对多个相关进程在执行次序上进行协调，使并发执行的进程之间能按照一定的规则共享系统资源，并能很好地相互合作，从而使程序的执行具有可再现性。

- 两种形式的制约关系：
 - 直接制约关系：相互协作，等待来自其他进程的信息，“同步”
 - 间接制约关系：进行竞争，独占分配到的部分或全部共享资源，“互斥”
- 进程同步：相互协调的几个进程在某些确定点上协调它们的工作，一个进程到达了这些点后，除非另一进程已完成了某些操作，否则就需要停下来等待这些操作的完成。
- 进程互斥：两个或两个以上的进程由于不能同时使用同一资源，只能一个进程使用完了另一个进程才能使用的现象。
- 临界资源：一次仅允许一个进程访问的资源。
- 临界区：临界段，在每个程序中，访问临界资源的那段程序。
- 进入区：在进入临界区之前，检查可否进入临界区的一段代码。如果可以进入临界区，通常设置相应“正在访问临界区”标志
- 退出区：用于将“正在访问临界区”标志清除。
- 剩余区：代码中的其余部分。
- 同步机制应遵循的准则：
 - 空闲让进
 - 忙则等待
 - 有限等待：应保证在有限时间内能进入临界区，以免陷入“死等”状态。
 - 让权等待：当进程不能进入临界区时，立即释放处理机以免陷入“忙等”状态。

(1) 硬件同步机制

1. 关中断：进程在临界区执行期间，计算机系统不响应中断。
2. Test-and-Set指令：为每个临界资源设置一个全局布尔变量lock，并赋初值false，表示资源空闲。在进入区利用TS进行检查：有进程在临界区时，重复检查；直到其它进程退出时，检查通过；

当进程进入后会将lock值设为TRUE表示资源正在被使用，使用完后会将其设为false。

3. Swap指令：为每个临界资源设置一个全局布尔变量lock，其初值为false，在每个进程中再利用一个局部变量key，初值为true，重复交换key与lock的值，直到key的值为false后代表可以占用临界资源。

- 缺点：
 - 等待要耗费CPU时间，，不能实现“让权等待”。
 - 可能导致“饥饿”，有的进程可能一直无法进入临界区。
 - 可能死锁。

(2) Dijkstra算法

- 信号量S的物理意义：
 - $S > 0$ 表示有S个资源可用
 - $S = 0$ 表示无资源可用
 - $S < 0$ 则 $|S|$ 表示S等待队列中的进程个数
- P原语P(S)：表示申请一个资源
- V原语V(S)：表示释放一个资源

利用mutex信号量实现互斥：

对P、V操作的讨论（重要！！）：


例题：

- 当进程A执行完毕后 $S=1$ ，此时进程B才可以运行。
- 当进程A与C执行结束后 S_1 与 S_2 才可以被进程B使用。

image-20200630202755514

(3) 经典进程同步问题（重点）

生产者/消费者问题

image-20200630203758064

-
- S_a 、 S_b 的初值限定了入库的数量，mutex限定了每次只能存入一种商品。
-

读者/写者问题

解法一：读者优先

- 一次只能有一个进程对readcount进行修改（mutex来实现），当readcount大于等于1时w互斥（但只在readcount=1时更改一次w的值），当readcount=0时释放w。

解法二：写优先

增加一个信号量，用于在写者到达后封锁后续的读者

哲学家就餐问题

5.进程通信（掌握PV操作实现的通信）

进程通信：指进程之间可直接以较高的效率传递较多数据的信息交换方式。

分类：

（1）共享存储器系统

- 基于共享数据结构的通信方式：诸进程公用某些数据结构，进程通过它们交换信息。
- 基于共享存储区的通信方式：在存储器中划出一块共享存储区,申请者把获得的共享存储分区连接到本进程上，此后可读写该分区

（2）消息传递系统：进程间的数据交换以消息为单位，程序员利用系统的通信原语实现通信


分为直接通信与间接通信。

（3）管道通信

管道：指用于连接一个读进程和一个写进程的文件

向管道提供输入的进程（称写进程），以字符流的形式将大量数据送入管道，而接受管道输出的进程（读进程）可从管道中接收数据。

- 消息传递通信的实现方式：
 - 直接消息传递系统
 - 信箱通信

image-20200630224839088

6.线程概念

- 线程：
 - 线程是进程内一个相对独立的、可调度的执行单元。
 - 进程中的一个运行实体，是一个CPU调度单位
 - 资源的拥有者还是进程
- 进程与线程对比：
 - 调度：线程上下文切换比进程上下文切换要快，同进程内的线程切换时间比进程短
 - 并发性：一个进程中的所有线程都能并发执行，不同进程中的线程也能并发执行。
 - 拥有资源：进程间相互独立，同一进程的各线程间资源共享。
 - 独立性：同一进程中的不同线程之间的独立性要比不同进程之间的独立性低得多。
 - 系统开销：在创建或撤销进程时，系统为之分配的资源明显大于线程创建或撤销时的开销。
 - 支持多处理机系统：多线程进程可以将一个进程中的多个线程分配到多个处理机上，使它们并行执行。

线程也有三个基本状态：执行、就绪、阻塞。

线程控制块TCB：通常含有线程标识符、寄存器、线程运行状态、优先级、线程专有存储区、信号屏蔽、堆栈指针。

7.线程实现

(1) **内核支持线程KST**: 在内核空间为每一个内核线程设置了一个线程控制块, 内核根据该控制块而感知某线程的存在, 并对其加以控制。

- 优点:
 - 在多处理系统中, 内核能够同时调度通过一进程中的多个线程并行执行。
 - 如果进程中一个线程被阻塞, 内核可以调度进程其它线程占有处理器, 也可以运行其它进程中的线程。
 - 线程的切换比较快, 切换开销小
 - 采用多线程技术, 可以提高系统的执行速度和效率。
- 缺点:
 - 线程切换时, 模式切换开销较大。

(2) **用户级线程ULT**: 在用户空间中实现的, 对线程的创建、撤销、同步与通信等功能, 都无需内核的支持。

- 优点:
 - 线程切换不需要转换到内核空间。
 - 调度算法可以是进程专用的。
 - 用户级进程的实现与OS平台无关, 所有应用程序都可以对之进行共享。
- 缺点:
 - 系统调用阻塞问题。当线程执行一个系统调用时, 进程内所有线程都会被阻塞。
 - 在单纯的用户级线程实现方式中, 进程中仅有一个线程能执行, 在该线程放弃CPU之前, 其它线程只能等待。

(3) 组合方式

用户级线程和内核支持线程两种方式结合。

第三章 处理机调度与死锁

1.调度层次与目标

- 层次:
 - 高级调度: 也称为作业调度或宏观调度, 从用户工作流程的角度, 一次提交的若干个流程, 其中每个程序按照进程调度。时间上通常是分钟、小时或天。
 - 中级调度: 涉及进程在内外存间的交换, 从存储器资源管理的角度来看, 把进程的部分或全部换出到外存上, 将当前进程所需部分换入到内存。
 - 低级调度: 也称进程调度、微观调度, 从处理机资源分配的角度来看, 处理机需要经常选择就绪进程或线程进入运行状态。
- 目标:
 - 处理机调度算法共同目标:
 - 资源利用率:
$$\text{CPU利用率} = \frac{\text{CPU有效工作时间}}{\text{CPU有效工作时间} + \text{CPU空闲等待时间}}$$
 - 公平性: 使进程都获得合理的CPU时间
 - 平衡性: 尽可能保持系统资源的平衡性
 - 策略强制执行: 对所指定的策略如安全策略必须准确执行
 - 批处理系统目标:
 - 平均周转时间短: 作业提交给系统到作业完成为止的时间段。
 - 系统吞吐量高: 单位时间内系统所完成的作业数
 - 处理机利用率高
 - 分时系统目标:

- 响应时间快
- 均衡性：指系统响应时间的快慢与用户所请求服务的复杂性相适应。
- 实时系统目标：
 - 截止时间的保障：指某任务开始执行的最迟时间。
 - 可预测性

2.作业与作业调度

- 作业：比程序更广泛的概念，程序+数据+作业说明书
- 作业控制块JC：是作业在系统中存在的标志
- **周转时间**：作业提交给系统到作业完成为止的时间段。
- **带权周转时间**：周转时间/进程服务时间
- 进程调度方法：
 - 非剥夺调度：某一进程被调度运行后，除非由于它自身的原因不能运行，否则一直运行下去。
 - 剥夺调度：当有比正在运行的进程优先级更高的进程就绪时，系统可强行剥夺正在运行进程的CPU，提供给具有更高优先级的进程使用。

先来先服务（FCFS）

- 每次调度是从就绪进程队列中选择一个最先进入该队列的进程分配处理机。
- 当前作业或进程占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）；在作业或进程唤醒后（如I/O完成），并不立即恢复执行，通常等到当前作业或进程出让CPU。
- 最简单的算法，有利于长作业而不利于短作业。

最短作业优先（SJF）

- 作业越短，优先级越高。
- 缺点：
 - 必须预知作业的运行时间
 - 对长作业非常不利
 - 未能依据作业的紧迫程度来划分执行的优先级

非抢占式：

抢占式：

最短剩余时间优先（SRT）

- 允许比当前进程剩余时间更短的进程来抢占

最高响应比优先（HRRN）

- 响应比 $R = \frac{(\text{等待时间} + \text{要求执行时间})}{\text{要求执行时间}} = \frac{\text{响应时间}}{\text{要求执行时间}}$

单道例题：

先来先去：

最短作业优先：

最高响应比：

多道例题（重要！！）：

- 两道批处理即可以两个作业同时调度，但一次仍只能运行一个作业。

时间片轮转算法 (RR)

- 系统中所有就绪进程按照FCFS原则排成队列。
- 每次调度时将CPU分派给队首进程，让其执行一个时间片。
- 在一个时间片结束时，发生时钟中断。调度程序暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。
- 进程可以未使用完一个时间片，就出让CPU（如阻塞、完成）。
- 响应时间的要求： $T(\text{响应时间}) = N(\text{进程数目}) * q(\text{时间片})$

基于优先级的调度算法

- 系统为每个进程设置一个优先数（对应一个优先级），把所有的就绪进程按优先级从大到小排序，调度时从就绪队列中选择优先级最高的进程投入运行，仅当占用CPU的进程运行结束或因某种原因不能继续运行时，系统才进行重新调度。
- 剥夺方式：
 - 非剥夺
 - 可剥夺
- 优先级类型：
 - 静态优先级：创建进程时就确定，直到进程终止前都不改变。
 - 动态优先级：在创建进程时赋予的优先级，在进程运行过程中可以自动改变，以便获得更好的调度性能。如等待时间延长优先级提高、每执行一个时间片就降低优先级等。

多级队列算法

- 根据作业或进程的性质或类型的不同，将就绪队列再分为若干个子队列。
- 每个作业固定归入一个队列。
- 各队列不同处理：不同队列可有不同的优先级、时间片长度、调度策略等。如：系统进程、用户交互进程、批处理进程等。

多级反馈队列算法

- 时间片轮转算法和优先级算法的综合和发展
- 优点：
 - 为提高系统吞吐量和缩短平均周转时间而照顾短进程
 - 为获得较好的I/O设备利用率和缩短响应时间而照顾I/O型进程
 - 不必估计进程的执行时间，动态调节
- 优先级分组法：组间可剥夺，组内不可剥夺（组内相同优先级则按FCFS处理）
- 基本思想：
 - 设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列1的优先级最高。每个队列执行时间片的长度也不同，规定优先级越低则时间片越长，如逐级加倍
 - 新进程进入内存后，先投入队列1的末尾，按FCFS算法调度；若按队列1一个时间片未能执行完，则降低投入到队列2的末尾，同样按FCFS算法调度；如此下去，降低到最后的队列，则按“时间片轮转”算法调度直到完成。
 - 仅当较高优先级的队列为空，才调度较低优先级的队列中的进程执行。如果进程执行时有新进程进入较高优先级的队列，则抢先执行新进程，并把被抢先的进程投入原队列的末尾
- 特点：
 - 短作业优先
 - 输入/输出进程优先

- 运算型进程有较长的时间片
- 采用了动态优先级, 使用珍贵资源 C P U 的进程优先级不断降低。 采用了可变时间片以适应不同进程对时间的要求, 运算型进程将获得较长的时间片

3.实时调度

- 实时系统:
 - 能够实现在指定或者确定的时间内完成系统功能和对外部或内部、同步或异步时间做出响应的系统
 - 在实时计算中, 系统的正确性不仅仅依赖于计算的逻辑结果, 而且依赖于结果产生的时间
- 实现实时调度基本条件:
 - 提供必要的信息: 就绪时间、开始截止时间和完成截止时间、处理时间、资源要求、优先级
 - 可调度的实时系统: 给定m个周期性事件, 事件i的周期为 P_i , 需要的处理时间为 C_i , 若处理机可调度则满足该条件: $\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$, 若多处理机则将1改为N。
 - 采用抢占式调度机制: 当一个优先权更高的任务到达时, 允许将当前任务暂时挂起, 而令高优先权任务立即投入运行, 可满足该实时任务对截止时间的要求。
 - 具有快速切换机制: 对外部中断的快速响应能力、快速的任務分派能力。
- 实时调度算法分类:
 - 非抢占式调度算法: 非抢占式轮转调度算法、非抢占式优先调度算法
 - 抢占式调度算法: 基于时钟中断的抢占式优先权调度算法、立即抢占(Immediate Preemption)的优先权调度算法
- 优先级反转问题: 高优先级进程因为资源被低优先级进程占据而被延迟或阻塞。
- 解决优先级反转: 优先级继承, 所有使用到高优先级进程所需资源的进程, 继承高优先级直到用完竞争资源, 再回到原来的优先级。

图中进程Task3继承了Task1的优先级, 使其不会被Task2抢占。

最低松弛度优先算法 (LLF)

- 实现该算法时要求系统中有一个按松弛度排序的实时任务就绪队列, 松弛度最低的任务排在队列最前面, 调度程序总是选择就绪队列中的队首任务执行
- 用于可抢占式调度
- 松弛度=必须完成时间-本身运行时间-当前时间

4.死锁

- 可重用性资源: 一次只能供一个进程安全地使用, 且不会由于使用而耗尽
 - 例子: 处理器, I/O通道, 主存和辅存, 设备, 文件、数据库、信号量等数据结构
- 可消费资源: 可以创建并且可以销毁的资源, 数目没有限制, 当一个进程得到一个可消费资源时, 这个资源就不再存在了
 - 例子: 中断, 消息, I/O缓冲区中的信息 (生产者进程创建, 消费者进程消耗)
- 可抢占性资源: 某进程在获得这类资源后, 该资源可以再被其它进程或系统抢占 (CPU、主存)
- 不可抢占性资源: 一旦系统把某资源分配给该进程后, 就不能将它强行收回, 只能在进程用完后自行释放 (刻录机、磁带机、打印机)
- 死锁**定义**: 如果一组进程中的每一个进程都在等待仅由该组进程中的其它进程才能引发的事件, 那么改组进程是死锁的。
- 产生死锁**原因**:
 - 竞争不可抢占性资源
 - 竞争可消耗资源

- 进程推进顺序不当
- 产生死锁**必要条件**:
 - 互斥条件: 指进程对所分配到的资源进行排它性使用, 即在一段时间内某资源只能由一个进程占有。如果此时还有其它进程申请该资源, 则它只能阻塞, 直至占有该资源的进程释放。
 - 请求和保持条件: 进程已经保持了至少一个资源, 但又提出了新的资源要求, 而该资源又已被其它进程占有, 此时请求进程阻塞, 但又对已经获得的其它资源保持不放。
 - 不可抢占条件: 进程已获得的资源, 在未使用完之前, 不能被剥夺, 只能在使用完时由自己释放。
 - 循环等待条件: 在发生死锁时, 必然存在一个进程-资源的封闭的环形链。即进程集合{P0, P1, P2, ..., Pn}中的P0正在等待一个P1占用的资源; P1正在等待P2占用的资源, ..., Pn正在等待已被P0占用的资源。
- 死锁**处理方法**:
 - 不允许出现死锁:
 - 预防死锁: 设置某些限制条件来破坏产生死锁的必要条件。
 - 避免死锁: 在资源动态分配过程中用某种方法防止系统进入不安全状态
 - 允许出现死锁:
 - 检测死锁
 - 解除死锁

5.死锁预防

概念: 在系统设计时确定资源分配算法, 保证不发生死锁 (破坏四个必要条件的一个或几个)

破坏请求和保持条件

- 协议一: 要求每个进程在运行前必须一次性申请它所要求的所有资源
- 协议二: 进程提出申请资源前必须释放已占有的一切资源
- 优点: 简单、易于实现
- 缺点:
 - 一个进程可能被阻塞很长时间, 等待资源, 发生饥饿
 - 资源严重浪费, 进程延迟运行

破坏不可抢占条件

- 方法一: OS可以剥夺一个进程占有的资源, 分配给其他进程 (只有当两个进程优先级相同时)
- 方法二: 一个已经保持了某些资源的进程, 当它再提出新的资源请求而不能立即得到满足时, 必须释放它已经保持的所有资源, 待以后需要时再重新申请
- 缺点: 实现复杂、代价大, 反复申请/释放资源, 系统开销大, 降低系统吞吐量

破坏循环等待条件

- 资源有序分配法: 把系统中所有资源编号, 进程在申请资源时必须严格按资源编号的递增次序进行, 否则操作系统不予分配
- 缺点:
 - 此方法要求资源类型序号相对稳定, 不便于添加新类型的设备
 - 易造成资源浪费, 类型序号的安排只能考虑一般作业的情况, 限制用户简单、自主地编程
 - 限制进程对资源的请求; 资源的排序占用系统开销

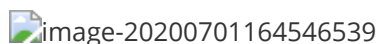
6.死锁避免

概念：在系统运行过程中，对进程发出的每一个系统能够满足的资源申请进行动态检查，并根据检查结果决定是否分配资源，若分配后系统可能发生死锁，则不予分配，否则予以分配

- 安全状态：如果存在一个由系统中所有进程构成的安全序列 P_1, \dots, P_n ，则系统处于安全状态
- 不安全状态：不存在一个安全序列。不安全状态不一定导致死锁，只是很可能死锁。
- 安全序列：一个进程序列 $\{P_1, \dots, P_n\}$ 是安全的，如果对于每一个进程 $P_i (1 \leq i \leq n)$ ，它以后尚需要的资源量不超过系统当前剩余资源量与所有进程 $P_j (j < i)$ 当前占有资源量之和，系统处于安全状态

银行家算法（重点）

银行家算法：



多种类资源情况；

利用银行家算法避免死锁（重点）：

数据结构：

(1) 可利用资源向量Available。含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。

(2) 最大需求矩阵Max。这是一个 $n \times m$ 的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果 $Max[i, j] = K$ ，则表示进程i需要 R_j 类资源的最大数目为K。

(3) 分配矩阵Allocation。一个 $n \times m$ 的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果 $Allocation[i, j] = K$ ，则表示进程i当前已分得 R_j 类资源的数目为K。

(4) 需求矩阵Need。一个 $n \times m$ 的矩阵，用以表示每一个进程尚需的各类资源数。如果 $Need[i, j] = K$ ，则表示进程i还需要 R_j 类资源K个，方能完成其任务。

(5) 请求向量Requesti是进程 P_i 的请求向量，如果 $Request_i[j] = K$ ，表示进程 P_i 需要K个 R_j 类型的资源。

当进程发出请求向量Request后，**系统执行检查**：

1. 如果 $Request_i[j] \leq Need[i, j]$ ，便转向步骤2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。
2. 如果 $Request_i[j] \leq Available[j]$ ，便转向步骤(3)；否则，表示尚无足够资源， P_i 须等待
3. 系统试探着把资源分配给进程 P_i ，并修改下面数据结构中的数值：

1. **Available[j] := Available[j] - Request_i[j] ;**

2. **Allocation[i, j] := Allocation[i, j] + Request_i[j] ;**

3. **Need[i, j] := Need[i, j] - Request_i[j] ;**

4. 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程 P_i ，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程 P_i 等待。

安全性检查（本质就是能否找到一个安全序列）：

(1) 设置两个向量：

① 工作向量Work: 它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时， $Work := Available$;

② Finish: 它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做 $Finish[i] := false$ ；当有足够资源分配给进程时，再令 $Finish[i] := true$ 。

(2) 从进程集合中找到一个能满足下述条件的进程：

① Finish [i] =false;

② Need [i,j] ≤Work [j] ; 若找到, 执行步骤(3), 否则, 执行步骤(4)。

(3) 当进程P_i获得资源后, 可顺利执行, 直至完成, 并释放出分配给它的资源, 故应执行:

Work [j] :=Work [i] +Allocation [i,j] ;

Finish [i] :=true;

go to step 2;


(4) 如果所有进程的Finish [i] =true都满足, 则表示系统处于安全状态; 否则, 系统处于不安全状态。

例题:

- 找到need<work的行, 并将alloc+work重新赋值给可以提供的值work
- request<need且request<available, 故先尝试分配 (括号内为分配后的)
- 分配后再来一次安全性检查, 如果安全则直接分配

练习 (掌握) :

- request>need

 image-20200701173205887

7.死锁的检测与解除

资源分配图:

资源分配图化简:

1. 找一个非孤立点进程结点且只有分配边, 去掉分配边, 将其变为孤立结点
2. 再把相应的资源分配给一个等待该资源的进程, 即将某进程的请求边变为分配边
3. 重复以上步骤, 若所有进程成为孤立结点, 称该图是可完全简化的, 否则称该图是不可完全简化的

死锁的**充分条件**: 当且仅当资源分配图是不可完全简化的。

- 死锁解除:
 - 重新启动
 - 撤销进程
 - 剥夺资源
 - 进程回退

##第四章 存储管理

1. 概念

存储管理: 对有限的内存块实施有效的管理

2.存储管理方法

(1) 单一连续分配

- 内存分为两个区域: 系统区, 用户区。应用程序装入到用户区, 可使用用户区全部空间。最简单, 适用于单用户、单任务的OS。
- 优点: 易于管理

- 缺点：对要求内存空间少的程序，造成内存浪费；程序全部装入，很少使用的程序部分也占用内存。

(2) 固定分区

- 把内存划分为若干个固定大小的连续分区。
- 分区方法：
 - 分区大小相等：处理多个类型相同的对象
 - 分区大小不等：根据程序大小，分配当前空闲、适当大小的分区。
- 优点：易于实现，开销小
- 缺点：内存碎片造成浪费；限制了并发执行的程序数目。

(3) 动态分区

- 在装入程序时按其初始要求分配，或在执行过程中通过系统调用进行分配或改变分区大小。
- 优点：没有内存碎片
- 缺点：有外碎片，如果大小不是任意的也可能出现内存碎片。

3. 基于顺序搜索的动态分区分配算法

- (1) 最先匹配法 (first-fit)：按分区的先后次序，从头查找，找到符合要求的第一个分区
- (2) 下次匹配法 (next-fit)：按分区的先后次序，从上次分配的分区起查找（到最后分区时再回到开头），找到符合要求的第一个分区
- (3) 最佳匹配法 (best-fit)：找到其大小与要求相差最小的空闲分区
- (4) 最坏匹配法 (worst-fit)：找到最大的空闲分区

例题：

- 先放一个再放下一个

-
- 解决分区碎片：
 - 多重分区：将程序装入分散存区中
 - 可重定位分配：将碎片集中（紧凑或拼接）
 - 将四个作业全部上移，留出134K的空闲空间。

4. 覆盖与交换

- 覆盖技术：一个作业的若干程序段，或几个作业的某些部分共享某一个存储空间。
 - 缺点：
 - 编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加编程复杂度
 - 从外存装入覆盖文件，以时间延长来换取空间节省
- 交换技术：系统将内存中某些进程暂时移到外存，把外存中某些进程换进内存，占据前者所占用的区域。
 - 原理：暂停执行内存中的进程，将整个进程的地址空间保存到外存的交换区中（换出swap out），而将外存中由阻塞变为就绪的进程的地址空间读入到内存中，并将该进程送到就绪队列（换入swap in）。
 - 优点：
 - 增加并发运行的程序数目，并且给用户提供适当的响应时间

- 编写程序时不影响程序结构
- 缺点：
 - 对换入和换出的控制增加处理机开销
 - 程序整个地址空间都进行传送，没有考虑执行过程中地址访问的统计特性
- 覆盖与交换比较：
 - 共同点：进程的程序和数据主要放在外存，当前需要执行的部分放在内存，内外存之间进行信息交换
 - 不同点：
 - 交换发生在进程或作业之间，而覆盖发生在同一进程或作业内
 - 与覆盖技术相比，交换技术不要求用户给出程序段之间的逻辑覆盖结构

5.分页存储管理

- 主存划分为大小相等的区域，称为块或内存块（物理页面，页框）
- 作业按照主存块大小分页，从0开始编制页号，页内地址是相对于0编址
- 连续的页存放在离散的块中，以页为单位进行分配，并按作业的页数多少来分配。逻辑上相邻的页，物理上不一定相邻
- 页表：一个数据结构，用来登记页号和块的对应关系和有关信息。
- 系统为每个进程建立一个页表，页表的长度和首地址存放在该进程的进程控制块（PCB）中。
- 页表包含：页号、块号、其它

两级页表（了解）：

- 快表：解决页地址变换中“访问主存=访页表+访主存”的问题。
- 把这种存放在快速存储器中的页表称为**快表**，把存放在内存中的页表称为**慢表**。
- 快表具有并行查询能力
 - 查联想表（访问一次主存）
 - 查页表（访问二次主存）

页地址映射计算：

快表问题：

6.分段存储管理

- 用户程序划分：程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名，且有一个段号。段号从0开始，每一段段内也从0开始编址，段内地址是连续的
- 内存划分：内存空间被动态的划分为若干个长度不相同的区域，称为物理段，每个物理段由起始地址和长度确定
- 内存分配：以段为单位分配内存，每一个段在内存中占据连续空间（内存随机分割，需要多少分配多少），但各段之间可以不连续存放
- 系统需要维护的数据结构：
 - 进程段表：描述组成进程地址空间的各段，可以是指向系统段表中表项的索引。每段有段基址 (base address)和段长度
 - 系统段表：系统内所有占用段
 - 空闲段表：内存中所有空闲段，可以结合到系统段表中

段地址映射：

- 段地址映射的数据结构有段表、段表首址指针和段表的长度。段表首址指针和段表长度存放在进程自己的PCB中。
- 每一进程有个段表，程序的每一个段在段表中占用一个表目。
- 段地址映射过程：
 - 程序地址字送入虚地址寄存器VR中
 - 取出段号S和段内位移W
 - 根据段表首址指针找到段表，查找段号为S的表目，得到该段的首地址
 - 把段首地址与段内位移相加，形成内存地址送入MR中，并以此地址访问内存

快表：与分页存储管理的快表一样。

- 优点：
 - 消除了内碎片
 - 通过请求分段存储管理方式提供了大量虚存
 - 允许动态增加段的长度
 - 便于动态装入和链接
 - 便于程序共享
 - 便于存储保护
- 缺点：
 - 进行地址变换和实现内存紧凑（靠拢）要花费处理机时间
 - 在辅存上管理可变长度的段比较困难

分段与分页的区别：

1. 段是依据程序的逻辑结构划分的，页是按内存线性空间物理划分的。
2. 段式技术中程序地址空间是二维的，分页技术中程序地址空间是一维的。
3. 段是面向用户的，页对用户而言是透明的。
4. 段长由用户决定，且各段的大小一般不相等，唯一的限制是最大长度。页长是由系统决定的，各页的长度必须相等
5. 段的共享比页的共享更容易

7.段页式存储管理

- 用户程序划分：按段式划分（对用户来讲，按段的逻辑关系进行划分；对系统讲，按页划分每一段）
- 内存划分：按页式存储管理方案
- 内存分配：以页为单位进行分配

地址映射：

- 段表：记录了每一段的页表始址和页表长度
- 页表：记录了逻辑页号与内存块号的对应关系（每一段有一个，一个程序可能有多个页表）
- 地址变换：先查段表，再查该段的页表

第五章 虚拟存储器

1.概念

- 局部性原理：在一较短时间内，程序的执行仅局限于某个部分，它所访问的存储空间也局限于某个区域。
- 虚拟存储器定义：指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。

- 实现方式：操作系统统一管理各级存储器；内存中只存放当前要执行的程序部分，其余的保存在外存上，OS根据需要随机地将需要的部分对换到内存执行
- 虚空间独立于实空间，指令中表示地址的位数越长，可寻址空间越大，主存+辅存 \neq 虚存
 - 虚空间逻辑大小=可寻址范围（例如32位操作系统可寻址范围是 $2^{32} = 4GB$ ）
 - 虚空间实际大小=内存+外存对换区
- 虚拟存储好处：
 - **大程序**：可在较小的可用内存中执行较大的用户程序
 - **大的用户空间**：提供给用户可用的虚拟内存空间通常大于物理内存
 - **并发**：可在内存中容纳更多程序并发执行
 - **易于开发**：与覆盖技术比较，不必影响编程时的程序结构
- 虚拟存储技术的特征：
 - 不连续性：物理内存分配的不连续，虚拟地址空间使用的不连续
 - 部分交换：与交换技术相比较，虚拟存储的调入和调出是对部分虚拟地址空间进行的；
 - 虚拟扩充：通过物理内存和快速外存相结合，提供大范围的虚拟地址空间（总容量不超过物理内存和外存交换区容量之和）
 - 多次对换：程序运行期间，分别在内、外存中的程序多次对换

2.请求分页存储管理方式

- 实现方法：作业运行时，只将当前的一部分装入内存其余的放在辅存，一旦发现访问的页不在主存中，则发出缺页中断，由o.s将其从辅存调入主存，如果内存无空块，则选择一个页淘汰。
- 缺页率=访问失败次数/访问总次数

页表结构：

缺页中断处理流程图：

页面置换算法

1. 先进先出页面算法（FIFO）：选择在内存中驻留时间最长的页并淘汰之
 2. 最近最久未使用置换算法（LRU, Least Recently Used）：淘汰没有使用的时间最长的页
 3. 最佳页面算法（OPT, Optimal）：淘汰以后不再需要的或最远的将来才会用到的页面（理论上的算法）
- 找后续访问序列中最远的一个数字（或不存在的数字）进行替换。
 - 4. 最不经常使用（LFU, Least Frequently Used）：选择访问次数最少的页面淘汰之

性能

- 影响缺页次数因素：
 - 分配给进程的物理块数
 - 页本身的大小
 - 程序的编制方法
 - 页面淘汰算法
- 常驻集：指虚拟页式管理中给进程分配的物理页面数目
- 抖动/颠簸：在虚存中，页面在内存与外存之间频繁调度，系统效率急剧下降，甚至导致系统崩溃。这种现象称为颠簸或抖动
- 请求分页优点：
 - 提供了虚存管理方式，作业地址空间不再受实存容量的限制
 - 更有效的利用了主存，方便于多道程序运行，方便了用户
- 缺点：

- 为处理缺页中断，增加了处理机时间的开销。用时间的代价换取了空间的扩大
- 可能因作业地址空间过大或程序数目过多等造成系统抖动；为此采取措施会增加的系统的复杂度

3.请求分段（考概念）

与请求分页管理方式类似，以分段为单位进行换入、换出。

请求分段中断处理的流程图：

###4.虚拟存储管理的其它问题

(1) 调入策略

- 请求调页：只调入发生缺页时所需的页面
- 预调页：在发生缺页需要调入某页时，一次调入该页以及相邻的几个页

(2) 分配策略

- 在虚拟段式管理中，如何对物理内存进行分配，可采用最佳适应、最先适应等。
- 在虚拟页式和段页式管理中，地址变换最后通过页表进行，因此不必考虑分配策略

(3) 清除策略

- 请求清除：该页被置换时才调出，把清除推迟到最后一刻
- 预清除：该页被置换之前就调出，因而可以成批调出多个页面

(4) 常驻集大小的动态调整策略——工作集策略

- 工作集：是一个进程执行过程中所访问页面的集合，可用一个二元函数 $W(t, \Delta)$ 表示
 - t 是执行时刻
 - Δ 是一个虚拟时间段，称为窗口大小(window size)，它采用"虚拟时间"单位(即阻塞时不计)，大致可以用执行的指令数目，或处理器执行时间来计算
 - 工作集是在 $[t - \Delta, t]$ 时间段内所访问的页面的集合

##第六章 输入输出系统

1.概述

- 设备分类：
 - 按传输速率分：
 - 低速设备
 - 中速设备
 - 高速设备
 - 按信息交换的单位分：
 - 字符设备
 - 块设备
 - 按资源管理方式分：
 - 独占型设备
 - 共享型设备
 - 虚拟设备
 - 按外部设备的从属关系分：
 - 系统设备
 - 用户设备

2.设备管理

- 设备管理目标：
 - 实现设备独立性
 - 提高设备利用率
 - 设备的统一管理
- 设备管理功能：
 - 监视系统中所有设备的状态
 - 设备的分配
 - I/O控制
- 设备控制块DCB：
- 设备控制器完成的工作：
 - （端口）地址译码
 - 接受主机发来的数据和控制信号，向主机发送数据和状态信号
 - 计算机的数字信号转换成机械部分能识别的模拟信号，或反之
 - 实现设备内部硬件缓冲、数据加工等提高性能或增强功能
- 设备接口：操作系统将命令写入控制器的接口寄存器（或接口缓冲区）中，以实现输入 / 输出，并从接口寄存器读取状态信息或结果信息
- 端口编址方法：
 - 内存映像编址（内存映像I/O模式）：分配给系统中所有端口的地址空间与内存的地址空间统一编址
 - 优点：
 - 凡是对存储器操作的指令都可对I/O端口操作
 - 不需要专门的I/O指令
 - I/O端口可占有较大的地址空间
 - 缺点：
 - 占用内存空间
 - I/O独立编址（I/O专用指令）：分配给系统中所有端口的地址空间完全独立，主机使用专门的I/O指令对端口进行操作
 - 优点：
 - 外部设备不占用内存的地址空间
 - 程序设计时易于区分是对内存操作还是对I/O端口操作
 - 缺点：
 - 对I/O端口操作的指令类型少，操作不灵活

3.I/O控制方法

1. 程序直接控制方法（循环查询I/O）：浪费大量CPU时间
2. I/O中断方式：CPU利用率大大提高，但数据缓冲寄存器每满一次都要中断一次，如果设备较多时，中断次数会很多，使CPU的计算时间大大减少。
3. DMA方式（直接存储器访问）
4. 通道方式：在CPU的控制下独立地执行通道程序，对外部设备的I/O操作进行控制，以实现内存与外设之间成批的数据交换
 - 字节多路通道
 - 数据选择通道
 - 数据多路通道

DMA方式与中断方式的区别

- 中断时机：
 - 中断方式是在数据缓冲寄存器满后，发中断请求，CPU进行中断处理
 - DMA方式则是在所要求传送的数据块全部传送结束时要求CPU进行中断处理（后处理）
- 数据传输：
 - 中断方式的数据传送由CPU控制完成
 - DMA方式是在DMA控制器的控制下不经过CPU控制完成的

4.I/O软件组成

5.缓冲技术

- 需要缓冲区的原因（重要）：
 - 缓和CPU与I/O设备间速度不匹配的矛盾
 - 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
 - 提高CPU和I/O设备之间的并行性
- 目标：用缓冲技术来匹配CPU与设备速度的差异和负荷的不均匀，从而提高处理机与外设的并行程度

(1) 单缓冲区

- 最简单的一种缓冲形式。当进程发出一I/O请求时，OS为之分配一缓冲区
- 对于输入：设备先将数据送入缓冲区，OS再将数据传给进程
- 对于输出：进程先将数据传入缓冲区，OS再将数据送出到设备

(2) 双缓冲区

- 原理：设置两个缓冲区buf1和buf2。读入数据时，首先输入设备向buf1填入数据，然后进程从buf1提取数据，在进程从buf1提取数据的同时。输入设备向buf2中填数据。当buf1取空时，进程又从buf2中提取数据，与此同时输入设备向buf1填数。如此交替使用两个缓冲区，使CPU和设备的并行操作的程度进一步提高。

(3) 环形缓冲区

- 在主存中分配一组大小相等的存储区作为缓冲区，并将这些缓冲区链接起来

(4) 缓冲池

- 缓冲池由内存中一组大小相等的缓冲区组成
 - 空(闲)缓冲区
 - 装满输入数据的缓冲区
 - 装满输出数据的缓冲区

6.磁盘驱动调度

- 磁盘分为两种：
 - 固定头磁盘
 - 移动头磁盘
- 磁盘访问过程：
 - 寻道：磁头定位到指定磁道
 - 旋转延迟：等待指定扇区从磁头下旋转经过
 - 数据传输：数据在磁盘与内存之间的实际传输
- 磁盘访问时间：

- 寻道时间 T_s ：启动磁臂时间 s 与磁头移动 n 条磁道所花费的时间之和： $T_s = m \times n + s$ ，其中 m 为一常数，与磁盘驱动器速度有关
- 旋转延迟时间 T_r ：不同磁盘类型旋转速度不同，该延迟一般也不同。
- 传输时间 T_t ：把数据从磁盘读出或向磁盘写入所经历的时间，大小与每次读写的字节数 b 和旋转速度有关： $T_t = \frac{b}{rN}$ ，其中 r 为磁盘每秒钟转数， N 为一条磁道上的字节数。

####磁盘调度算法

(1) 先来先服务 (FCFS, First-Come, First Served)

- 按访问请求到达的先后次序服务
- 优点：简单、公平
- 缺点：效率不高，相邻两次请求可能会造成最内到最外的柱面寻道，使磁头反复移动，增加了服务时间，对机械也不利

(2) 最短寻道时间优先 (SSTF, Shortest Seek Time First)

- 优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先
- 优点：改善了磁盘平均服务时间
- 缺点：造成某些访问请求长期等待得不到服务

(3) 扫描算法 (电梯算法, SCAN)

- 具体做法：当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复

(4) 循环扫描调度算法 (CSCAN)

- 扫描算法的改进，总是从0号柱面开始向里扫描。移动臂到达最后一个柱面后，立即带动读写磁头快速返回到0号柱面。返回时不为任何的等待访问者服务。返回后可再次进行扫描（即单向扫描）

例题：

先来先服务：

最短寻道时间优先：

扫描算法：

循环扫描调度算法：

- 提高磁盘I/O速度方法：
 - 磁盘高速缓存 (Disk Cache) :
 - 在内存中开辟一个单独的存储空间作为磁盘高速缓存
 - 把所有未利用的内存空间变为一个缓冲池，供分页系统和磁盘I/O共享
 - 优化数据分布：
 - 优化物理块的分布：物理块连续分布、增加物理块大小等可以减少磁头的移动
 - 优化索引结点的分布：将索引结点放在中间位置、将磁道分组，每组都有索引结点等。
 - 其它方法：
 - 提前读：在访问文件时经常是顺序访问，因此在读当前块时可以提前读出下一块
 - 延迟写：修改缓存中的数据后一般应立即写回磁盘，但该盘块可能还会被修改，立即写回会带来很大的开销
 - 虚拟盘：利用内存仿真磁盘，又称RAM盘

7.设备分配

- 设备分配方法：
 - 静态分配：在作业级进行的，当一个作业运行之前由系统一次分配满足需要的全部设备，这些设备一直为该作业占用，直到作业撤消。这种分配不会出现死锁，但设备的利用效率较低。
 - 动态分配：在进程运行的过程中进行的，当进程需要使用设备时，通过系统调用命令向系统提出设备请求，系统按一定的分配策略给进程分配所需设备，一旦使用完毕立即释放。显然这种分配方式有利于提高设备的使用效率，但会出现死锁，这是应力求避免的。
- 设备服务算法：
 - 先请求先服务
 - 优先级高的优先服务
- 根据设备特性分为：独享设备、共享设备和虚拟设备
- 对应三种分配技术：
 - 独享分配：独占型设备有行打印机，键盘，显示器。磁带机可作为独占设备，也可作为共享设备。即当进程申请独占设备时，系统把设备分配给这个进程，直到进程释放设备。
 - 共享分配：磁盘，磁带和磁鼓。对这类设备的分配是采用动态分配的方式进行的，当一个进程要请求某个设备时，系统按照某种算法立即分配相应的设备给请求者，请求者使用完后立即释放。
 - 虚拟分配
- **虚拟分配**：为提高计算机系统的效率，提出了在高速共享设备上模拟低速设备功能的技术，称为虚拟设备技术
 - 实现过程：当用户(或进程)申请独占设备时。系统给它分配共享设备的一部分存储空间。当程序要与设备交换信息时，系统就把要交换的信息存放在这部分存储空间。在适当的时候再将存储空间的信息传输到相应的设备上去处理
 - 共享设备中代替独占设备的那部分存储空间和相应的控制结构称为虚拟设备，并把对这类设备的分配称作虚拟分配

8.SPOOLing系统（假脱机）

- 在联机情况下实现的同时外围操作称为SPOOLing，也称为假脱机操作
- SPOOLing系统组成：
 - 输入井和输出井
 - 输入缓冲区和输出缓冲区
 - 输入进程和输出进程
- SPOOLing工作原理：
 - 作业执行前预先将程序和数据输入到输入井中
 - 作业运行后，使用数据时，从输入井中取出
 - 作业执行不必直接启动外设输出数据，只需将这些数据写入输出井中
 - 作业全部运行完毕，再由外设输出全部数据和信息
- 好处：
 - 实现了对作业输入、组织调度和输出的统一管理
 - 使外设CPU直接控制下，与CPU并行工作（假脱机）
- 特点：
 - 提高了I/O速度
 - 将独占设备改造为共享设备
 - 实现了虚拟设备功能

1.概念

- **文件**：是赋名的信息(数据)项的集合，是赋名有关联的信息单位(记录)的集合
- **文件名**：文件的标识符号，一个用来标识文件的有限长度的字符串
- 文件的组成：
 - 文件体：文件本身的信息
 - 文件说明：文件存储和管理信息；如：文件名、文件内部标识、文件存储地址、访问权限、访问时间等
- **文件系统**：操作系统中负责管理相关文件信息的软件机构
- 文件系统管理的对象：文件、目录、磁盘空间
- 文件系统的组成：被管理的文件+与文件管理相关的软件+实施文件管理所需的数据结构
- 文件类型
 - 性质和用途分类：
 - 系统文件：只能通过系统调用为用户服务
 - 库文件：允许用户调用但不允许用户修改
 - 用户文件：用户委托操作系统保存的文件
 - 文件的保护方式分类：只读文件、读写文件、不保护文件
 - 文件的信息流向：输入文件、输出文件、输入输出文件
 - 文件的逻辑结构分类：流式文件、记录式文件
 - UNIX系统对文件的分类：
 - 普通文件：包含的是用户的信息
 - 目录文件：管理文件系统的系统文件
 - 特殊文件：设备文件，外部设备也看作文件
- 文件系统要实现的功能模块：
 - 文件的分块存储：与外存的存储块相配合
 - I/O缓冲和调度：性能优化
 - 文件定位：在外存上查找文件的各个存储块
 - 外存存储空间管理：如分配和释放。主要针对可改写的外存如磁盘
 - 外存设备访问和控制：包括由设备驱动程序支持的各种基本文件系统如硬盘，软盘，CD ROM等
- 文件系统接口：
 - 命令接口。这是指作为用户与文件系统交互的接口。用户可通过键盘终端键入命令，取得文件系统的服务
 - 程序接口：这是指作为用户程序与文件系统的接口。用户程序可通过系统调用来取得文件系统的服务

2.文件逻辑结构

- 无结构的流式文件：
 - 基本信息单位是字节或字，其长度是所含字节的数量
 - 优点：节省存储空间，无需额外的说明和控制信息
- 有结构的记录式文件：由若干个记录组成，文件中的记录可按顺序编号为记录1，记录2，.....，记录n
 - 定长记录文件
 - 变长记录文件

(1) 连续文件

- 一个逻辑文件的信息存放在存储器上的相邻物理块中，该文件为连续文件，这样结构称为连续结构
- 优点：
 - 顺序存取速度快，所需的磁盘寻道次数和寻道时间最少。
 - 简单，支持顺序存取和随机存取
- 缺点：
 - 不便记录的增删操作，一般只能在末端进行
 - 在建立连续结构文件时，要求用户给出文件的最大长度，以便系统分配足够的存储空间，但这个有时候难以办到

(2) 链接结构

- 在每个物理块中设置一指针，指向该文件的下一个物理块号，文件的末尾块存放结束标记“NULL”
- 优点：
 - 文件可以动态扩充，也不必事先提出文件的最大长度
 - 由于不连续分配，不存在外部碎片问题，所以不会造成几块连续区域的浪费
 - 有利于文件插入和删除
- 缺点：
 - 存取速度慢，不适于随机存取，只适合顺序存取
 - 每块设置链接字破坏物理信息的完整性
 - 链接指针占用一定的空间
- 文件分配表（FAT）：将盘块中的链接字按盘块号的顺序集中起来，构成盘文件映射表/文件分配表 FAT

(3) 索引文件

- 为文件建立一张索引表，每个记录设置一个表项。索引表按记录关键字排序，本身是顺序文件。在对索引文件进行检索的时候，首先按照顺序文件检索方法查找索引表，从中找到相关表项，然后直接访问该记录
- 优点：
 - 保持了链接结构的优点，又解决了其缺点
 - 即能顺序存取，又能随机存取
 - 满足了文件动态增长、插入删除的要求
 - 能充分利用外存空间
- 缺点：
 - 索引表本身带来了系统开销，如：内外存空间，存取时间

(4) 文件映照

- 直接索引：可根据给定的记录键值，直接获得指定记录的物理地址
- 哈希文件：记录位置由哈希函数确定。检索时给出记录编号，通过哈希函数计算出该记录在文件中的相对位置。

结构比较

- 从查询时间看：连续文件最快, 索引文件和文件映照次之, 串联文件最慢
- 从空间开销来看：连续文件不需要额外的空间开销; 串联文件的每个物理块中需要存放链接字; 文件映照需存放文件映照表; 索引文件为每个文件建立一张索引表
- 从适宜设备和存取方法来看：连续文件可用于磁带和磁盘; 串联文件、索引文件只适用于磁盘; 串联文件只适合顺序存取; 而索引文件和磁盘上的连续文件, 除了能进行顺序存取外, 也能实现随机存取
- 从文件增删来看：连续文件不能动态增长, 其他三种都可较容易实现文件的动态改变

文件存取三种方法：

- 顺序存取法：严格按文件信息单位排列的顺序依次存取
- 直接存取法：也称随即存取法，每次存取操作时必须先确定存取的位置。对流式文件或定长记录的文件比较容易确定存取位置；对不定长的记录式文件比较麻烦
- 按键存取法：文件的组织按照逻辑记录中的某个数据项的内容来存放，根据记录内容进行存取

3.文件目录

- 文件目录组织目标：能方便而迅速地目录进行检索，从而准确地找到所需文件
- 文件控制块FCB：是操作系统为管理文件而设置的数据结构，存放了为管理文件所需的所有有关信息
 - 基本信息类：文件名、文件物理位置、文件逻辑结构、文件的物理结构
 - 存取控制信息类
 - 使用信息类
- 文件目录：把所有的FCB组织在一起，就构成了文件目录，即文件控制块的有序集合
- 目录项：构成文件目录的项目（一个FCB就是目录表中的一个目录项）
- 目录文件：为了实现对文件目录的管理，通常将文件目录以文件的形式保存在外存，这个文件就叫目录文件

(1) 一级目录

- 整个目录组织是一个线性结构，系统中的所有文件都建立在一张目录表中。它主要用于单用户操作系统
- 优点：结构简单、清晰, 便于维护和查找
- 缺点：查找速度慢、不允许重名、不便于实现文件共享

(2) 二级目录

- 在根目录下，每个用户对应一个目录（第二级目录）；在用户目录下是该用户的文件，而不再有下级目录。适用于多用户系统，各用户可有自己的专用目录。
- 优点：
 - 提高了检索目录的速度
 - 在不同的用户目录中，可以使用相同的文件名
 - 不同用户还可使用不同的文件名来访问系统中的同一个共享文件
- 缺点：缺乏灵活性，特别是不能反映现实世界中多层次的关系

(3) 多级目录

- 或称为树状目录(tree-like)。在文件数目较多时，便于系统和用户将文件分散管理
- 多级目录结构由根目录和各级目录组成，为管理上的方便，除根目录外，其它各级目录均以文件的形式组成目录文件
- 各级目录文件称中间结点，用方框表示。数据文件称为叶结点，用圆圈表示
- 路径名：在多级目录结构中一个文件的唯一标识不再是文件名，而是从根结点开始，经过一个或多个中间结点，到达某个叶结点的一条路径。称这条路径为文件的路径名，它是文件的唯一标识
- 当前目录：当前使用的文件所在的目录指定为工作目录(或称当前目录)
- 绝对路径名：指由根目录开始的路径名
- 相对路径名：指从当前工作目录开始的路径名
- 索引结点：改进的多级目录
- 增加和删除目录：
 - 不删除非空目录：当目录(文件)不空时，不能将其删除，而为了删除一个非空目录，必须先删除目录中的所有文件，使之先成为空目录，后再予以删除。

- 可删除非空目录：当要删除一目录时，如果在该目录中还包含有文件，则目录中的所有文件和子目录也同时被删除。
- 目录查询技术：
 - 线性检索法：利用所提供的文件名顺序查找。
 - Hash方法：系统利用用户提供的文件名，将它变换为文件目录的索引值，再利用该索引值到目录中去查找。

###4.文件共享与保护

- 文件共享：一个或部分文件由事先规定的某些用户共同使用
- 文件保护：文件不得被未经文件主授权的任何用户使用
- 文件共享目的：
 - 节省存储空间
 - 进程间通过文件交换信息
- 基于索引结点的文件别名：也称为**硬链接**，通过多个文件名链接(link)到同一个索引结点，可建立同一个文件的多个彼此平等的别名。别名的数目记录在索引结点的链接计数中，若其减至0，则文件被删除。
- 基于符号链接的文件别名：**软链接**，它是一种特殊类型的文件，其内容是到另一个目录或文件路径的链接。建立符号链接文件，并不影响原文件，实际上它们各是一个文件。可以建立任意的别名关系，甚至原文件是在其他计算机上。

文件保护（了解）

- (1) 访问控制矩阵：一维代表所有用户，一维代表系统中的所有文件
- (2) 存取控制表
- (3) 用户权限表
- (4) 口令实现：用户为自己的每一个文件设置一个口令，存放在文件的FCB中。任何用户要存取该文件，都必须提供和FCB中一致的口令，才有权存取
- (5) 密钥实现

###5.文件存储空间管理

- 存储空间分配方法：
 - 预分配：创建时(这时已知文件长度)一次分配指定的存储空间，如文件复制时的目标文件
 - 动态分配：需要存储空间时才分配（创建时无法确定文件长度），如写入数据到文件
- 文件存储单位：簇（cluster）
 - 大小：大到能容纳整个文件，小到一个外存存储块
 - 文件卷与簇大小的关系：
 - 文件卷容量越大，若簇的总数保持不变即簇编号所需位数保持不变，则簇越大
 - 文件卷容量越大，若簇大小不变，则簇总数越多，相应簇编号所需位数越多
- 文件存储分配数据结构：连续分配、链式分配、索引分配

(1) 空白文件

- 辅存上的一片连续的空闲区，可视为一个空白文件，系统设置一张空白文件目录来记录辅存上所有空闲块的信息。每个表目存放一个空白文件的信息，包括该空白文件第一个空闲块号、空闲块个数、该文件所有空闲块号等信息
- 适用于连续文件结构
- 缺点：

- 如果文件太大, 那么在空白文件目录中将没有合适的空白文件能分配给它。
- 经过多次分配和回收, 空白文件目录中的小空白文件越来越多, 很难分配出去, 形成碎片

(2) 空白块链

- 把所有的空闲块链接在一起, 形成一个空闲块链表。释放和分配空白块都可以在链首处进行
- 优点:
 - 可实现不连续分配
 - 节省了存储开销
- 缺点
 - 系统开销大
 - 对于大型文件系统, 空闲链将会太长

(3) 位示图 (Bit Map)

- 用一串二进制位反映磁盘空间分配使用情况, 每个物理块对应一位, 分配物理块为1, 否则为0
- 申请物理块时, 可以在位示图中查找为0的位, 返回对应物理块号; 归还时; 将对应位转置0

位示图下的**磁盘分配**:

1. 顺序扫描位示图, 从中找出一个或一组其值为“0”的二进制位(“0”表示空闲)
2. 将所找到的一个或一组二进制位, 转换成与之相应的盘块号。假定找到的其值为“0”的二进制位, 位于位示的第*i*行、第*j*列, 则其相应的盘块号应按下式计算: $b = n(i-1) + j$, 其中*n*代表每行的位数
3. 修改位示图, 令 $map[i, j] = 1$

磁盘的回收:

1. 将回收盘块的盘块号转换成位示图中的行号和列号。转换公式为: $i = (b-1) \text{DIV } n + 1$, $j = (b-1) \text{MOD } n + 1$
2. 修改位示图。令 $map[i, j] = 0$

例题:

-
- 磁盘分区: 把一个物理磁盘的存储空间划分为几个相互独立的部分, 称为“分区”
 - 文件卷: 在同一个文件卷中使用同一份管理数据进行文件分配和外存空闲空间管理, 而在不同的文件卷中使用相互独立的管理数据
 - 一个文件不能分散存放在多个文件卷中, 其最大长度不超过所在文件卷的容量
 - 通常一个文件卷只能存放在一个物理外设上
 - 格式化: 在一个文件卷上建立文件系统
 - 建立并初始化用于进行文件分配和外存空闲空间管理的管理数据
 - 通常, 进行格式化操作使得一个文件卷上原有的文件都被删除