第一章 绪论

一.操作系统的概念

1..什么是操作系统：在计算机硬件之上会覆盖一层软件，以方便用户使用计算机硬件。

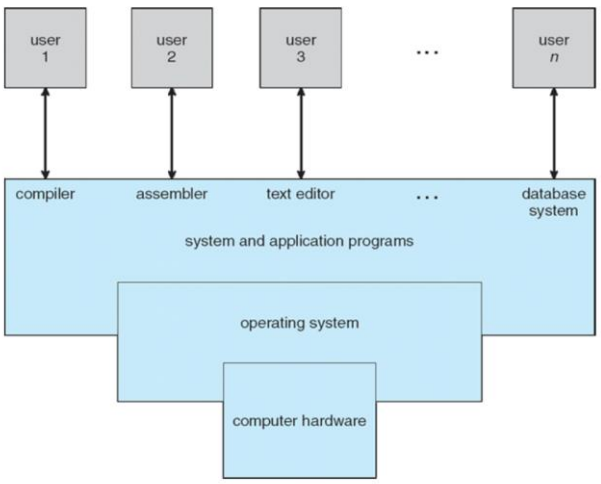
如果在裸机之上覆盖一层 I/O 设备管理软件，就能使用户较方便地使用外部设备；如果在其上再覆盖一层文件管理软件，用户就很容易存取系统文件和用户文件；每覆盖一层新的**软件**，就构造了一台功能更强的虚拟机器。通过 OS，计算机能提供种类更多，质量更高的服务。

2.定义：**把操作系统定义为用以控制和管理计算机系统资源，方便用户使用的程序和数据结构的集合。**

3.计算机系统可以分为四个组件：硬件、操作系统、应用程序以及用户

从这张图里也可以看到，操作系统位于应用程序和电脑硬件之间，是二者的中介。

硬件包括：CPU、内存、输入输出设备、系统总线



操作系统是**硬件之上的第一层软件，是对硬件的首次扩充，又是其他软件运行的基础**

4.操作系统的作用：

（1）计算机硬件、软件资源的管理者（管理进程、存储器、文件系统、I/O 设备）

（2）为用户使用计算机硬件、软件提供接口：

主要提供两种接口：命令接口（允许用户输入命令调用资源——终端）、程序接口（通常为系统调用，允许用户程序从用户态切换为管态，为应用程序使用硬件资源提供接口）

（3）操作系统是直接与硬件相邻的第一层软件，它是由大量的系统程序（中断处理程序、I/O 驱动程序）和数据结构（FCB、PCB）集合成的

二、操作系统的发展

1.发展历程:

(1) 真空管，无操作系统（1946~50 年代）:

a.人工操作

b.I/O：纸袋或者卡片，用户手工装入

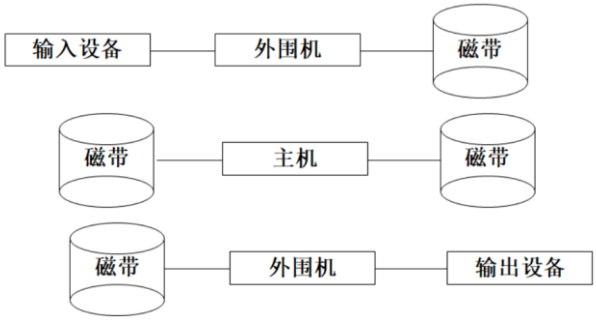
c.缺点：用户独占全机，资源利用率低;

CPU 与 I/O 速度不匹配，CPU 利用率低

(2) 单道批处理系统（50~60 年代）:

a.批处理：把类似的任务放在一起，一次处理一批任务

b.脱机 I/O：在外围机的帮助下，以磁带作为输入中介，避免 CPU 等待用户



c.主机在计算一个程序的时候，外围机可以提前处理下一个程序的输入和上一个程序的输出，这样就实现了并行。

d.运行特征:

1.顺序性：磁带上的各道作业程序是顺序地进入内存，完成顺序与他们进入内存的顺序相同。

2.单道性：内存中仅有一道程序运行

3.自动性：批处理系统会自动按照顺序一个一个地处理作业（自动为作业排序）

e.优点：减少了 CPU 的空闲等待时间，提高了 CPU 和 IO 设备的使用效率，提高了吞吐量

缺点：CPU 和 IO 设备忙先不均（如果 CPU 需要执行 IO 操作，发起 IO 请求后高速 CPU 就要等待慢速的 IO，无法实现真正的并行）

(3) 多道批处理系统（60~70 年代）

a.特点:

多道——内存中同时存放几个作业，每个作业都处于开始点和结束点之间，多个作业共享 CPU、内存等资源;

当 CPU 执行作业 1 时，若发出 I/O 请求，操作系统会保存其上下文并将其挂起，然后调度作业 2 来继续使用 CPU。I/O 完成后，设备会发出中断信号，操作系统响应中断并将作业 1 设置为'就绪'。作业 1 是否立即恢复执行，取决于调度策略。当轮到它时，它将恢复原先状态，继续执行。

b.通道：一种负责外部设备与内存之间信息传输的专用部件

中断：主机接收到外界的信号时，停止原来的工作，转去处理外来事件，处理完成后又回到原来工作点继续工作。

c.互斥和同步机制：

互斥：多个线程/进程不能同时访问同一临界资源

同步：多个线程/进程按照特定的顺序协调运行

d.特征：

1.多道性：内存中同时主流多道程序并发执行

2.无序性：作业的完成顺序与它进入内存的顺序之间没有严格的对应关系

3.调度性：作业调度、进程调度

（4）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（允许多个应用程序同时存在于内存中，同时对用户提供服务）

a. 分配方式：时间片轮换（把 CPU 时间分成若干个大小相等的时间单位，每个终端用户获得一定个数的时间片开始运行，当时间片到，该用户程序暂停挂起，等待下一次分配到时间片）

b. 特点：

a) 多路性（同时性）：众多联机用户可以同时使用同一台计算机；

b) 独占性：各终端用户感觉到自己独占了计算机；（因为响应很快）

c) 交互性：用户与计算机之间可进行“会话”；

d) 及时性：用户的请求能在很短时间内获得响应。

（5）实时系统：在短于分时系统一个时间片内响应（有严格确定的时间限制）

（6）并行系统：多处理器系统（具有多个 CPU，处理器共享内存和时钟）

（7）分布式系统

（8）网络操作系统：在通常操作系统功能的基础上**提供网络通信和网络服务功能**的操作系统

分布式系统下，一个任务由多台物理计算机共同完成，而网络系统下一个任务通常只由一台计算机完成。

（9）嵌入式系统

三、操作系统的功能

1.操作系统的五大功能：进程管理、内存管理、设备管理、文件管理、用户接口

2.进程管理：

（1）进程控制：创建、删除、挂起、改变进程优先级

（2）进程调度：作业和进程的运行切换

（3）进程同步：协调并发进程之间的推进步骤

（4）进程通信：进程之间的信息交换

3.内存管理：存储分配与回收、存储保护、虚拟内存

4.设备管理：为用户分配 IO 设备、缓冲管理、虚拟设备、为应用程序提供统一的接口

5.文件管理：文件存储空间的管理（内碎片外碎片）、目录管理（文件检索）、文件存取控制（防止未授权或者恶意篡改文件）

6.用户接口：为用户提供命令级接口（命令行）和程序级接口（系统调用）

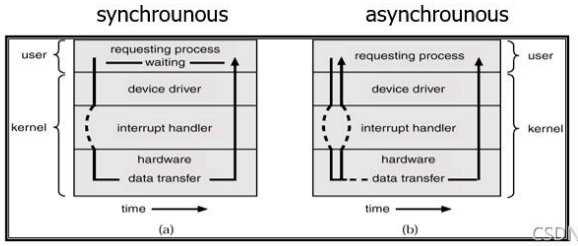
四、操作系统的特点

1.操作系统的四大特征：并发、共享、虚拟、异步
2.并发性：在同一段时间内处理多个作业（在每个时间点只有一个作业执行）——注意与并行（同一时刻有多个事件执行）的区别
3.共享性：多个进程共享有限的计算机资源，操作系统要对系统资源进行合理的分配和使用（共享包括：互斥共享和同时访问）
4.虚拟性：虚拟式操作系统管理系统资源的重要手段，可以提高资源利用率
5.异步性：进程的执行顺序和执行时间不可预知——异步性可能会导致程序运行出现与时间按相关的错误，操作系统要保证执行结果正确（同步）

第二章 计算机系统结构

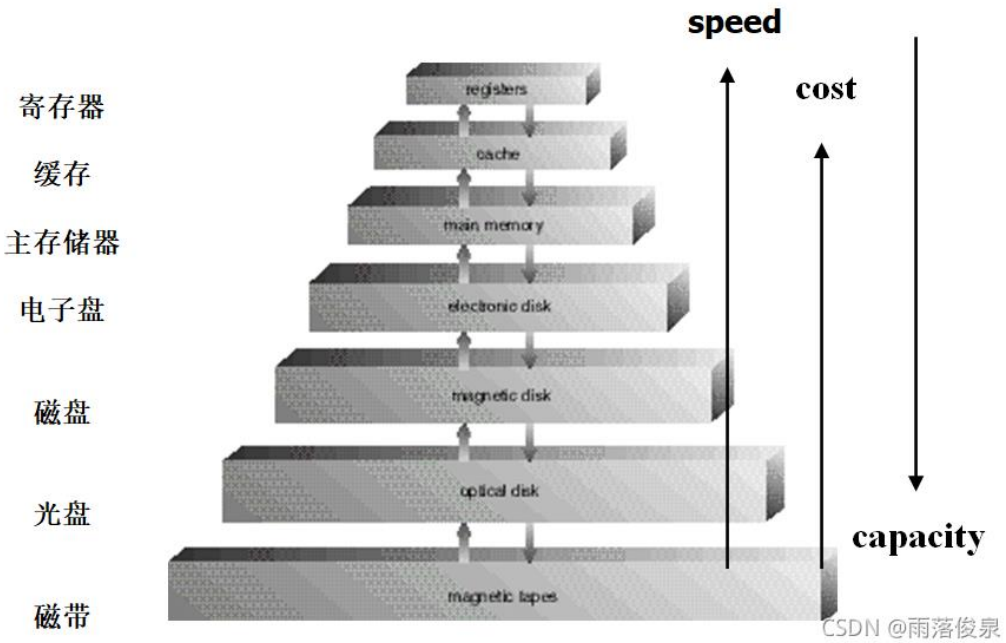
一、计算机系统操作

1. 中断机制：包括硬件中断和软件中断
   a) 硬件中断：设备控制器利用中断通知 CPU 它已经完成某个操作
   b) 软件中断：也称为“陷阱”，包括异常和系统调用
   现代操作系统是中断驱动的，中断会将 CPU 控制权转移到中断服务程序，有通用的程序检测是否有中断，对于不同的中断会有相应的代码来进行处理。
   c) 中断向量是中断服务程序的入口，将所有的中断向量集中放在一起，形成中断向量表，中断向量表存放在内存的低地址空间
   d) I/O 中断：同步（当 I/O 操作开始时，系统会等待 I/O 操作结束才执行其他的操作）和异步（当 I/O 操作开始时，系统不会等待，而是直接去执行其他的操作）



   e) 用设备状态表管理 I/O 设备：存储每个设备的状态、等待队列等；当发生中断时，操作系统会查阅设备状态表（DST），确定发出中断的设备，更新 DST 并进行调度
2. DMA 操作：直接内存访问（Direct Memery Access）：设备控制器在缓存和内存之间直接进行数据的块传输，而不经过 CPU 的控制，只是在数据传输完成时向 CPU 发送一个中断。

二、存储结构



1. 主存（Main Memery）：主要包括随机访问存储器（Random Access Memery,RAM）
2. 二级存储：对主存的扩展，主要是磁盘
3. Cache：缓存，暂时保存最近访问过的数据，CPU 优先访问 Cache，如果没有命中，则访问 MM。如果还是没有命中，则访问进行 IO 操作访问外存（磁盘 disk）

三、硬件保护

1. 对计算机资源（文件、数据）进行保护
2. 两状态操作：在计算机硬件添加模式位，表示当前模式
   a) 用户态（User mode）：执行用户程序
   b) 管态（Monitor mode）：当出现中断（需要进行 IO 操作或者调用系统资源）时——系统调用，硬件会切换到管态。
   使用特权指令告诉系统需要进行状态切换
3. IO 保护：所有 IO 操作都是特权指令，只能通过系统调用在管态下执行
4. 内存保护：基本寄存器（记录某个程序在内存中的起始地址）和限制寄存器（记录某个寄存器在内存中的最大长度），只能通过系统调用的特权指令加载修改，对用户不可见。
5. CPU 保护：设置定时器（管态），防止程序陷入死循环或不返回 CPU 执行权给操作系统定时器时间到，产生中断，CPU 控制权返回给 OS

第三章 操作系统结构

一、系统调用：由操作系统提供给进程的一个接口，属于软中断。

系统调用是操作系统暴露给用户态程序访问**硬件资源或核心功能**的唯一合法途径

a) 系统调用的处理机制：陷入或异常处理机制

b) 系统调用的过程：

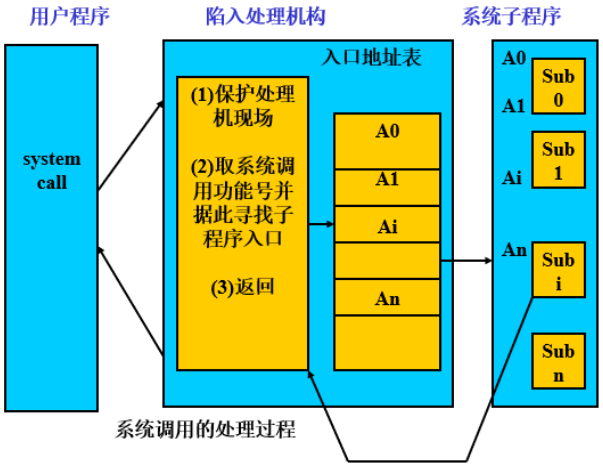
i. 当用户（应用程序）使用系统调用时，会产生一条相应的指令

ii. CPU 在执行到这条指令时，发出有关的信号给陷入处理机制（中断处理机制）

iii. 在处理系统调用之前，陷入处理机构首先保存处理机现场（PC、用户栈指针、通用寄存器、用户自定义参数等）

iv. 处理机制在收到 CPU 信号后，启动相应的程序完成该系统调用的请求（需求操作）——通过入口地址表找到相应系统调用的系统程序，并执行该系统程序

v. 系统调用处理完成后，恢复处理机现场感，从而使得用户程序继续执行



系统调用的处理过程

c) 系统调用的类型：

i. 进程控制

ii. 文件管理

iii. 设备管理

iv. 信息维护

v. 通信

vi. 保护

二、系统程序：指运行在用户态的程序，作为操作系统的一部分，提供用户与操作系统内核之间的接口和服务支持。它们为用户程序的开发、运行、维护以及系统管理提供便利，是操作系统功能在用户空间的延伸。

三、操作系统结构

a) 层次化的结构设计：

i. 分层的基本原则：每一层都使用其底层所提供的功能和服务，以便于系统调式和验证。——保证不出现双向依赖关系

ii. 优点：

1. 低层和高层可以分别实现（便于扩展）

2. 高层的错误不会影响到低层，便于调试、利用功能的增删改

iii. 缺点：

1. 系统中所有进程的控制转移和通信等任务全部交给系统的核心去管理，需要花费一定的代价。

四、微内核：

a) 定义：通过划分系统程序和用户程序，把所有不必要的部件移出内核，形成一个小内核，微内核提供最少量的进程管理、存储管理以及通信功能

b) 操作系统的两大组成部分：

i. 运行在核心态的内核

ii. 运行在用户态的进程层

c) 微内核只保留了传统 OS 的任务管理、进程控制和内存管理三个组成部分，其余部分（文件系统、设备管理、保护等）均被移出内核作为运行在用户态的系统程序实现

d) 微内核的优点：

i. 易于扩充、易于移植

ii. 提供多种操作环境、便于实现分布计算

iii. 在低层可以通过内核的网络传送到远程服务器上

e) 微内核的缺点：消息传递方式开销增加，响应变慢

五、模块（宏内核）：内核提供核心服务，其他服务在内核运行时可动态加载，类似于分层，但是更加灵活，任何模块可以彼此调用，也类似于微内核，主模块只有核心功能，并知道如何加载其他模块和如何让模块进行通信(擦破做系统的给想功能被划分为不同的模块，在运行时采用动态加载)

第四章 进程

一、进程的概念

a) 进程的五个状态：

i. 新建 new：在创建进程

ii. 就绪 ready：进程等待分配处理器,已经获得了除处理器以外的所有资源

iii. 运行 running：指令在执行

iv. (阻塞)等待 waiting：进程等待某些事件发生

v. 终止 terminated：进程执行完毕

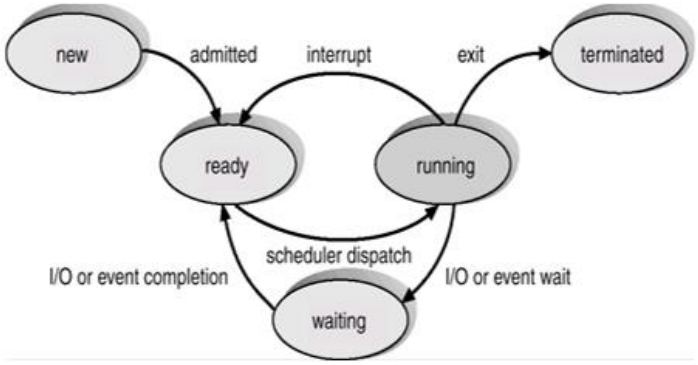
b) 进程的状态转换：

i. 就绪状态-->执行状态：一个进程被进程调度程序选中

ii. 执行状态-->阻塞状态：请求并等待某个事件发生

iii. 执行状态-->就绪状态：时间片用完或在抢占式调度中有更高优先级的进程变为就绪状态

iv. 阻塞状态-->就绪状态：进程因为等待的某个条件发生而被唤醒



c) 进程的七状态模型：

i. 挂起：当系统内存不足的时候，由操作系统或者用户手动地将某些未运行的进程（就绪进程/阻塞进程）从内存中移出到磁盘，缓解内存压力，等待重新被调入内存执行的过程。

ii. 相较于五状态模型，七状态模型增加了就绪挂起和阻塞挂起两种状态

d) 进程控制块(Process Control Block——PCB)：

i. 定义：PCB 是一个专门用来记录进程的外部特征，描述进程的运动变化过程的专门的数据结构（以表的形式组织）

ii. 每个进程都有唯一的 PCB 与之对应

iii. PCB 经常被系统访问（调度程序、资源分配、中断处理），因此需要常驻内存。

iv. PCB 包含的内容：

1. 进程标识符（PID）
2. 父进程标识符（PPID）
3. 进程状态
4. CPU 寄存器（记录当进程被暂停时 CPU 各个寄存器的内容，方便快速地恢复到运行状态）
5. PC
6. IO 状态

二、进程调度

a) 进程调度地的不同目的：
   i. 多道程序的进程调度：最大化 CPU 利用率
   ii. 分时系统的进程调度：快速实现 CPU 在不同进程之间的切换

b) 调度队列：
   i. 作业队列：所有进入系统但是仍未进入内存的程序的集合（处于外存磁盘上）
   ii. 就绪队列：在内存中，就绪并等待分配 CPU 时间片执行的进程的集合
   iii. 设备队列：等待某一 IO 设备的进程队列（针对 IO 设备而言）

c) 调度程序：
   i. 长程调度（作业调度）：选择可以进入就绪队列的进程（从外存到内存）——长程调度控制了多道程序的“道”
   ii. 短程调度（CPU 调度）：选择下一个被执行的进程（从就绪队列中选择就绪进程并分配时间片）
   iii. 中程调度（交换）：为了缓解内存紧张，将内存中处于阻塞 or 就绪状态的进程从内存换到外存磁盘上（阻塞挂起 or 就绪挂起），同时，中程调度还负责将这些进程换回到内存——降低多道程序的“道”

d) 上下文切换：当 CPU 从一个进程切换到另一个进程时，系统会将就进程的上下文保存在 PCB 中，并且加载新进程上下文

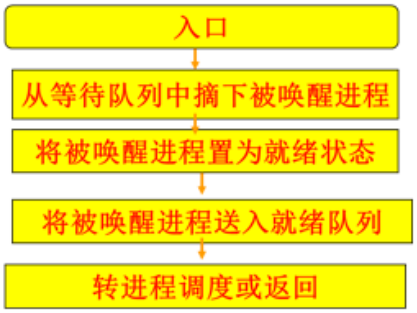
三、对进程的操作：

a) 进程创建(process creation)：作业调度程序选中外存上的某个作业后，把改作业装入内存，并分配必要的资源，创建进程(PCB)，加入就绪队列。
   i. 父进程创建子进程：构成进程树
      1. 资源共享的方式：
         a) 父子进程共享所有的资源
         b) 子进程共享父进程资源的子集
         c) 父子进程无共享
      2. 父子进程同时存在时的两种执行方式（单核）：
         a) 父进程与子进程并发执行
         b) 父进程等待子进程的执行结果
      3. 子进程的地址空间：
         a) 子进程是父进程的复制品，具有和父进程完全相同的程序和数据
         b) 子进程加载另一个新的程序
   ii. 创建进程的过程：
      1. 申请空白的 PCB
      2. 为新建立的进程分配资源
      3. 初始化 PCB（填入 PID、初始化处理机状态、初始化控制信息（设置进程状态为就绪））
      4. 将新进程插入到就绪队列

b) 进程终止(process termination)：当进程执行完成后通过系统调用请求操作系统删除自身，进程终止，所有进程资源会被 OS 释放
   i. 僵尸进程：子进程已经终止，但父进程没有通过 wait()获取其退出状态，导致该子进程的进程表项（包括 PID）还留在内核中。
   ii. 孤儿进程：父进程没有调用 wait()就终止，其子进程成为孤儿进程

c) 进程阻塞(prodess blocking)：处在运行状态的进程，因为等待某个事件的发生而不能继续运行时，将调用阻塞原语，把进程设置为阻塞状态，加入某事件的等待队列，释放 CPU

d) 进程唤醒(process wakeup)：当阻塞进程所等待的事件发生时，该进程将被唤醒，PCB 状态被设置为就绪状态，加入就绪队列



四、协同进程与独立进程：

a) 独立进程不会影响另一个进程的执行或被另一个进程执行影响
b) 协同进程可能影响另一个进程的执行或被另一个进程执行影响
c) 协同进程的优点：
   i. **Information sharing 信息共享**
   ii. **Computation speed-up 加速运算**
   iii. **Modularity 模块化**
   iv. **Convenience 方便**

五、进程间的通信：

进程间的通信通常有两种方式：共享内存（部分经过内核）和消息传递（必须全部经过内核）

a) 共享内存：在两进程之间建立共享存储区，进程通过读写共享区实现信息交换

b) 消息传递：

i. 若两进程需要通信，则需要建立通信连接交换信息

ii. 消息传递的两种方式：

1. 直接通信：显式命名 PID，通过 send 和 receive 语句实现消息传递

send(P,message)：向进程P发信息

receive(Q,message)：从进程Q收消息

2. 间接通信：消息导向至信箱并从信箱中接受

a) 操作：

i. 创建信箱

ii. 通过信箱发送和接受消息

iii. 销毁信箱

send(A, message) – send a message to mailbox A

receive(A, message) – receive a message from mailbox A

mailbox A

第五章 线程

一、进程与线程

a) 进程的基本属性——并发执行的基本单位：

i. 进程是拥有资源（虚存地址、内存空间）的独立单位

ii. 进程是可独立调度和分配的基本单位

b) 进程存在的缺点：**由于进程是一个资源的拥有者，因而在进程创建、撤销、调度切换时，系统需要付出较大的时空开销。进程的数目不宜过多，进程切换频率不宜过高，限制了并发程度。**

c) 线程的基本思想：将进程的两个基本属性分开——对于拥有资源的基本单位，不对其进行频繁的切换；对于调度的基本单位，不作为资源拥有的单位，“轻装上阵”

d) 引入线程的目的：简化线程间的通信，以小的开销来提高进程内的并发程度。

Eg.一个应用程序可能需要执行多个不同的任务（在某些情况下，一个应用程序可能需要执行多个相似的任务（循环、网页访问）），这时引入线程避免为每个任务创建进程导致的较大的系统开销

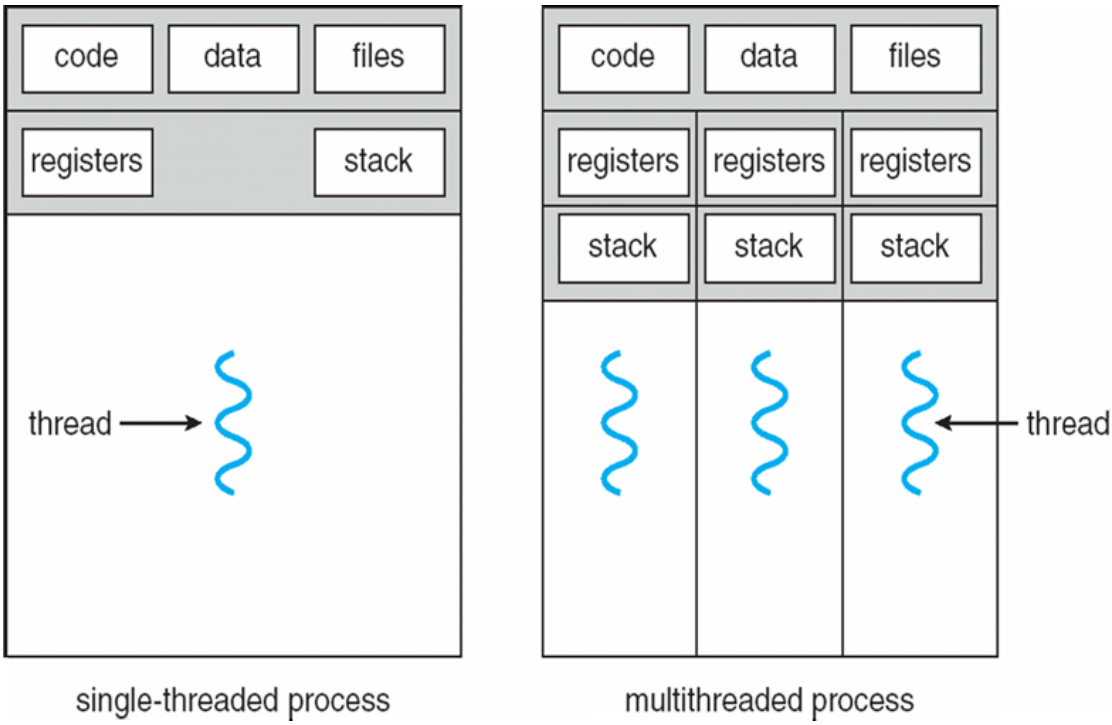
二、线程的概念

a) 线程是 CPU 调度的基本单位，而进程只作为其他资源分配的单位（也成为轻型进程）

b) 线程只拥有必不可少的资源（线程状态、程序计数器、寄存器上下文）；线程也有**就绪**、**阻塞**和**执行**三种状态，从属于一个进程的线程共享进程拥有的全部资源，可

以并发执行（从属于不同进程的线程也能够并发执行）

c) 线程的优点：**减小并发执行的时间和空间开销（线程的创建、退出和调度），因此容许在系统中建立更多的线程来提高并发程度；同进程内的线程可以在不通过内核的情况下直接通信。（一个线程和他的对等线程共享代码段、数据段和操作系统资源，线程切换时只需要切换寄存器和栈，所以会比进程切换快）**



三、进程与线程的比较

a) 并发性：不仅进程之间可以并发，在同一个进程中的多个线程之间也能并发执行——更有效地使用系统资源和提高系统的吞吐量
b) 资源：进程是拥有资源的独立单位
c) 系统开销：操作进程的系统开销远大于操作线程的系统开销
d) 通信：线程之间的通信可以直接读写进程数据段（如全局变量）；
e) 调度：线程上下文切换更快

四、内核线程与用户线程

a) 内核线程：由内核的内部需求进行创建和撤销，用来执行特定的函数——时间片分给线程
b) 用户线程：由用户级线程库进行管理的线程，线程库提供对线程的创建和管理的支持，无需内核支持（用户线程的调度由线程库管理，不需要 OS 的支持）——时间片分给进程
c) 调度单位：用户线程的调度以进程为单位进行，在采用时间片轮转调度算法时，每个进程分配相同的时间片。对内核级线程，每个线程分配时间片。
d) 多线程模型：
   i. 多对一：多个用户级线程映射到一个内核线程（任意时刻只有一个线程可以访问内核），如果一个线程发起系统调用而阻塞，则整个进程阻塞
   ii. 一对一：每个用户级线程映射进内核线程（有一个内核线程与之对应）；每创

建一个用户级线程需要船舰一个相应的内核级线程，带来额外的系统开销

iii. 多对多

第六章 CPU 调度

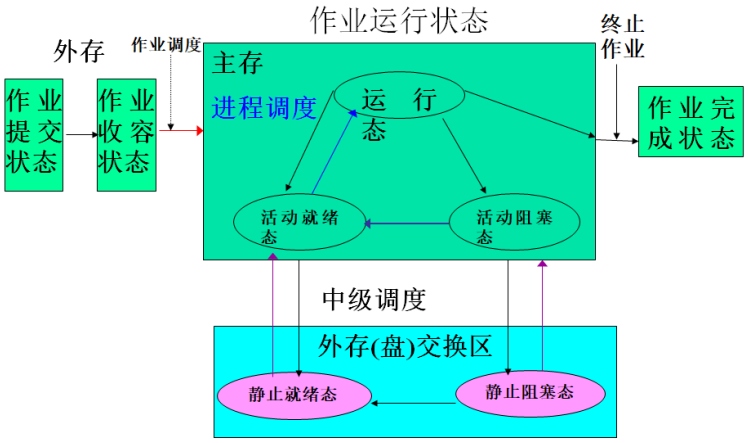
一、CPU 调度的概述：

CPU 调度分为长程调度、中程调度和短程调度：

（1） 长程调度：从提交到系统的作业中选择一个进入内存，为其创建进程，分配除 CPU 之外的其他资源，加入就绪队列，等待时间片的分配

（2） 短程调度：决定就绪队列中的哪个进程获得时间片，从就绪状态转化为执行状态（由分派程序执行把处理机分配给改进程的操作）

（3） 中程调度：当内存区域不够用时，将处于阻塞或者就绪队列的进程从内存中换出到磁盘上，缓解内存资源紧张



二、CPU 调度的基本概念

a) 定义：当 CPU 空闲时(当进程离开 running 状态时)，OS 就选择**内存中**的某个就绪进程，并给其分配 CPU

b) 时机：

i. 从运行到等待

ii. 从运行到就绪

iii. 从等待到就绪

iv. 终止运行

其中，1 和 4 属于非抢占式调度，2 和 3 属于抢占式调度（判断依据：进程是否是主动让出 CPU）

c) 抢占式和非抢占式调度：

i. 非抢占式调度：将处理机分配给某个进程后，便让其一直执行，直到该进程完成或者发生某件事而被阻塞时，才会将 CPU 分配给其他进程，不允许其他进程抢占已经分配出去的处理机

ii. 抢占式调度：允许调度程序根据某个原则，去停止某个正在执行的进程，将

处理机重新分配给另一个进程。

iii. 抢占原则：

1. 时间片原则：当时间片用完后，暂停该进程的执行重新进行调度
2. 优先权原则：当具有较高优先级的进程进入就绪队列时，终止正在执行的进程，将 CPU 分配给具有更高优先级的进程，使之执行。
3. 短作业优先原则

三、调度准则

a) 吞吐量：单位时间内运行完的进程数
b) 周转时间：进程从提交到运行结束的全部时间
c) 等待时间：进程在就读队列中等待调度的时间总和
d) 响应时间：从进程提出请求到首次被相应的时间段（从提交到系统到第一次获得 CPU）
e) 带权周转时间：作业周转时间与 CPU 时间的比值（一个作业从提交到完成的总时间是其真正占用 CPU 时间的几倍）

四、调度算法

a) 先来先服务(FCFS)：按照作业到达后备作业队利（或进程进入就绪队列）的先后次序来选择作业（或进程）——非抢占式——有利于长作业，不利于短作业；有利于 CPU 型作业，不利于 IO 型作业
b) 短作业优先(SJF)：选择执行时间最短的作业优先调度——非抢占式或抢占式：取决于新来的进程是否会抢占当前未完成进程的 CPU 资源——最优的（有利于系统减少平均周转时间，提高吞吐量）——但是可能出现饥饿
c) 优先级调度：每个进程都有自己的优先级（静态分配或动态分配——根据进程等到时间的延长提高其优先级——避免了饥饿的现象）——非抢占式或抢占式
d) 时间片轮转法（RR）：每个进程分配到小单位的 CPU 时间（时间片），时间片用完后，该进程的 CPU 资源将被抢占并被插入到就绪队列末尾
e) 多级队列(MQS)：按照进程的属性来分类，每类进程组成一个就绪队列，每个进程固定地处于某一个队列——每个队列都有自己的调度算法
f) 多级反馈队列（MFQS）：存在多个就绪队列，具有不同的优先级，各自按时间片轮转法调度；当一个进程配抢占 CPU，则对其进行降低优先级操作，当一个进程在低优先级就绪队列中等待时间过长时，则适当提高其优先级（就绪队列的优先级可以按照作业的长短进行划分）——高优先级的就绪队列分配到的时间片较短——尽快相应用户的请求，完成进程的切换

五、多机调度（多处理器调度）

a) 对称式多处理器：每个处理器决定自己的调度方案。每个 CPU 自己到公共就绪队列去取进程来执行。这需保证多个 CPU 对公共就绪队列的互斥访问
b) 非对称多处理器：仅一个处理器能处理系统数据结构，这就减轻了对数据的共享需求（在非对称多处理器结构中，只有一个处理器（通常称为主处理器或主核）负责所有的调度、系统调用和 I/O 处理，其余处理器（称为从处理器）只能在主处理器的控制下执行用户进程。）

第七章 同步

进程的同步保证多个进程（或线程）在并发执行时对共享资源进行协调和有序地访问的机制

一、背景

多个进程对共享数据的并发访问可能导致数据的不一致性，要保证数据的一致性，就需要保证并发进程的正确执行顺序（同步机制保证多个进程对共享数据的互斥访问）

1、进程的分类：

a) 协作进程

b) 独立进程

2、进程间资源访问的冲突：

a) 共享变量的修改冲突（不同进程对共享变量的读-改-写顺序不当：eg.进程一读，暂停，进程二读、改、写，进程一改、写——导致执行结果不符合预期）

b) 操作顺序冲突：本质是操作之间的先后依赖关系（eg.生产者消费者问题中，消费者在生产者写数据之前就从缓冲区读取了数据——数据为空或非法）

二、临界区问题（The Critical-Section Problem）

a) 临界区：进程中访问临界资源的一段代码

b) 如何实现对临界资源的互斥访问：各进程互斥地访问自己的临界区（互斥地执行临界区的代码）——每个进程都有一段可能修改共享变量/更新表/写文件的代码，当一个进程执行在临界区时，其他进程都不能进入临界区（执行修改共享资源的代码段）

c) 临界区的执行在时间上是互斥的，进程必须请求允许进入临界区

d) 临界区问题的解决方案应该满足的条件：

i. 互斥(Mutual Exclusion)：当进程 P 在临界区内执行时，禁止其他进程进入自己的临界区

ii. 有空让进(Progress)：当没有进程在临界区时，应该允许某个想进入的进程立即进入，而不能无谓等待。

iii. 有限等待：对一个想进入临界区的进程，必须保证它在有限次后一定能进入（即不能被无限期地饿死）

iv. 让权等待：不能进入临界区的进程，应该释放 CPU（转换到阻塞状态）

三、用信号量(Semaphores)实现进程的同步与互斥

a) 进程的同步：不同之间有执行上的先后依赖关系（Q 进程必须等待 P 进程完成才能继续执行）

进程的互斥：不同进程之间竞争地使用某个共享变量，为了保证程序执行结果的正确性，必须保证每时刻只有一个进程对该共享变量进行访问

b) 信号量：

i. 二元数组 S(s,q)：其中 s 表示某类资源的可用个数，q 表示当前请求该资源但是被阻塞的队列

ii. **整型变量 s 表示系统中某类资源的数目，大于零时表示当前可用资源数，小于零时，其绝对值表示因请求该类资源而被阻塞的资源数**

iii. **信号量的值仅由 P 操作（wait 操作）和 V 操作（signal 操作）进行改变**

1. P/V 操作：

```
P (S):
        S--;
        if  S< 0 do block;
V(S):
        S++;
        if S<=0 then wakeup;
```

注意：P 操作先对信号量减一，再判断是否大于 0（先登记申请，再判断是否有资源，否则进行 V 操作时不知道是否有进程需要申请资源，无法确定是否要执行唤醒操作）

```
semaphore s;
P(s) {
    s.value--;
    if (s.value < 0) {
        add this process to list s.L
        block
    }
}
V(s) {
    s.value++;
    if (s.value <= 0) {
        remove a process P from list s.L
        wakeup(P);
    }
}
```

```
typedef struct{
  int value;
  struct process *L;
}semaphore
```

P/V 操作被硬件从底层封装成原子操作——不可中断（保证了互斥）

c) 利用信号量实现互斥：为临界资源设置一个信号量（表示该资源的可用数量），其初值设置为 1；每个进程的临界区代码都要放置在对该操作的 PV 操作之间

d) 利用信号量实现同步：为每个前驱关系设置一个同步信号量（明确前驱关系，先释放+1，再使用-1），其初值设置为 0（只有执行完前驱进程的 V 操作，后继进程才能从阻塞状态脱离，继续执行）

信号量实现互斥：同一个进程对共享资源有一套完整的 PV 操作（避免其他进程访问）；　　信号量实现同步：两个进程对前驱关系信号量构成一套 PV 操作（先 V 操作生产资源，再 P 操作消耗资源）

e) 经典的同步问题：

i. 哲学家就餐问题：

1. 问题描述：5 个哲学家围绕一张圆桌而坐，桌子上放着 5 支筷子，每两个哲学家之间放一支；哲学家的动作包括思考和进餐，进餐时需要同时拿起他左边和右边的两支筷子，思考时则同时将两支筷子放回原处。如何保证哲学家们的动作有序进行？如：不出现相邻者同时要求进餐；不

出现有人永远拿不到筷子；

2. 信号量：筷子是临界资源，设立一个信号量数组表示筷子

3.
```
Repeat
    思考；
    取chopStick[i];//拿起左手边筷子
    取chopStick[(i+1) mod 5]；//拿起右手边筷子
    进食；
    放chopStick[i];
    放chopStick[(i+1) mod 5]；
Until false;
```

4.
可能会出现死锁，五个哲学家每人拿起了他左边的筷子,会导致五个筷子都被占用，当他们试图拿右边的筷子时，会“无限等待”

解决办法

> 最多允许四个哲学家同时就坐
>
> 同时拿起两根筷子
>
> 非对称解决——奇偶交替拿

ii. 生产者-消费者问题：

1. 问题描述：若干进程通过有限的共享缓冲区交换数据。其中，"生产者"进程不断写入，而"消费者"进程不断读出；共享缓冲区共有 N 个；任何时刻只能有一个进程可对共享缓冲区进行操作。
2. 信号量设置：
   a) 两个同步信号量(empty=N 和 full=0)：消费必须等到生产之后进行，生产必须在缓存不满的情况下进行
   b) 一个互斥信号量（mutex=1）：每一时刻只能由一个进程访问缓存区

3.

Producer	Consumer
P(empty); P(mutex); //进入区	P(full); P(mutex); //进入区
one unit --> buffer;	one unit <-- buffer;
V(mutex); V(full); //退出区	V(mutex); V(empty); //退出区

注意：在既有同步信号量又有互斥信号量的情况下，应该先 P 同步信号量，只有当同步信号量有可用资源时，才去 P 互斥信号量（获取互斥锁），否则的话先获取互斥锁则会导致其他进程阻塞无法修改同步信号量，整个系统发生死锁

iii. 读者写者问题：

1. 问题描述：对共享资源的读写操作，任一时刻“写者”最多只允许一个，而“读者”则允许多个：“读－写”互斥，“写－写”互斥，"读－读"允许

f) PV 操作的总结：

i. PV 操作必须成对出现：

1. 当为互斥操作时，PV 操作在同一个进程中
2. 当为同步操作时，PV 操作处于不同进程中（信号量初值设为 0，前驱进程 V 操作，后继进程进行 P 操作）

ii. **对于前后相连的两个 P(S1)和 P(S2) ，顺序是至关重要的:同步 P 操作应该放在互斥 P 操作前,而两个 V 操作顺序则无关紧要**

四、信号量集

五、管程

a) 管程是管理进程间同步的机制，它保证进程互斥地访问共享变量，并方便地阻塞和唤醒进程。

b) 管程的基本思想是**把信号量及其操作原语封装在一个对象内部**。即：将共享变量以及对共享变量能够进行的所有操作集中在一个模块中。

c) 定义：**管程是关于共享资源的数据结构及一组针对该资源的操作过程所构成的软件模块。（面向对象设计）**

第八章 死锁

一、死锁产生的原因和必要条件

a) 死锁的定义：计算机系统中多道程序并发执行时，两个或两个以上的进程由于竞争资源而造成的一种互相等待的现象，如果没有外力作用，这些进程永远不能再向前推进

b) 如果一个进程要使用 OS 管理的资源，需现象 OS 提出申请，如果有可用资源，系统才进行分配

c) 产生死锁的原因：

i. 竞争资源引起死锁

ii. 进程推进顺序不当引起死锁（多道程序中，并发执行的进程推进顺序不可预知）

二、死锁的特征：

a) 互斥：一次只用一个进程可以使用一个资源（各个进程之间对资源的使用是互斥的）

b) 占有并等待：一个进程至少占有一个资源并等待另一个资源，而该资源被另一个进程占有

c) 不可抢占：一个资源只用当持有它的进程使用完毕后才能由进程本身释放

d) 循环等待：等待资源的进程之间形成环（我等你的资源，你等他的资源。他等我的资源）

三、资源分配图

a) 定义：资源分配图是两类节点和两类边的集合

i. 两类顶点:圆形节点指的是进程 p

方框形节点指的是资源 R（方框中黑点的个数代表该类资源的实例个数）

ii. 两类边:由进程指向资源——请求边 P->R

由资源指向进程——分配边 R->P

b) 判断死锁：

i. 如果图中没有环：不会出现死锁

ii. 如果图中有环：

1. 如果每一种资源类型只有一个实例，那么发生死锁
2. 如果每种资源类型有多个实例，可能发生死锁

四、处理死锁的方法：

a) 鸵鸟算法：视而不见
b) 预防死锁：抑制死锁的必要条件（互斥、占用并等待、非抢占、循环等待）
c) 避免死锁：在资源分配过程中，用某种方式防止系统进入不安全状态
d) 检测与解除死锁：检测出死锁的产生，然后采用某种措施解除

注意：死锁预防和死锁避免的区别：

死锁预防指的是在所有进程执行前由操作系统统一规定的一系列规范来避免可能出现的思绪哦情况（要求进程在执行前一次性申请全部的资源），属于静态策略；而避免死锁指的是在进程执行过程当中，当进程需要申请资源时，由操作系统检查对该资源的申请石佛偶可能导致系统陷入到不安全的状态（银行家算法），如果可能使系统陷入到不安全状态，则拒绝进程对该资源的请求，是一种动态策略

五、银行家算法：

a) 安全状态：系统的某种状态，在这种状态下，系统能够按照某种顺序来为各个今晨分配其所需的资源，直至最大需求，使得每个进程都可以顺序地一个一个完成
b) 通过绘制表格 Work.need,allocaction,finish 来完成对安全序列的寻找
c) 通常，安全序列不唯一

六、死锁恢复：死锁发生后，如何处理死锁

a) 进程终止：

i. 终止所有进程
ii. 一次终止一个进程直到死锁环消失
iii. 选择终止顺序：进程的优先级；进程运行了多长时间，还需要多长时间

b) 资源抢占

i. 逐步抢占某个被牺牲进程的资源，直到打破死锁
ii. 回退：返回到安全状态，然后重新开始进程

总结：死锁的预防、死锁的避免、死锁的检测、死锁的恢复是死锁处理的四种策略，其中死锁预防和死锁避免需要进行区分，死锁预防是由操作系体强制设定的静态策略，死锁避免是在资源分配前动态检测的；检测方法包括资源分配图和银行家算法，其中资源分配图适合于

每类资源只用一个实体的时候，这时，一旦分配图中出现环，则一定有死锁发生；当一类资源中有多个实体时，就应该采用银行家算法查找安全序列。

第九章 内存管理

一、背景知识

a) 程序必须加载进入内存才能执行
b) 作业队列（输入队列）：磁盘上等待进入内存并执行的作业的集合
c) 用户程序的执行过程：
   i. 经过编译程序将源代码编译成若干个目标模块；
   ii. 其次通过链接程序将编译好的目标模块以及所需的库函数链接在一起，形成完成的装入程序
   iii. 最后将装入程序加载至内存等待 CPU 执行

   注意：程序的编译不是在内存中执行的，而是在磁盘上执行的
d) 指令和数据绑定到内存地址可以在三个不同的阶段发生
   i. 编译时期：如果内存位置已知，可以编译生成绝对编码（硬编码）
   ii. 装入时期：如果编译时期内存位置未知（取决于进程装入内存的地址），在编译时生成可重定位编码
   iii. 执行时期：如果进程在内存中的位置可以发生改变（中程调度），那么此时内存地址绑定需要延迟到程序执行时才能进行，这时需要硬件对地址映射的支持（需要基址和限长寄存器）

二、物理地址与逻辑地址

a) 逻辑地址（虚拟地址）：由 CPU 产生
b) 物理地址：内存设备所读入的地址（真实存在的地址）
c) 地址重定位：将程序装入到与地址空间不一致的物理空间所引起的一些列地址变换过程
   i. 动态地址重定位：在程序执行过程中，在 CPU 访问内存之前,将要访问的程序或数据地址转换成内存地址. 动态重定位依靠硬件地址变换机构完成（在程序的编译、链接、装入阶段使用的都是逻辑地址，不会绑定具体的物理内存地址；**在程序运行时**，由操作系统+硬件（通常是 MMU，内存管理单元）**将逻辑地址转换为物理地址**）

三、交换(Swapping)

a) 一个进程可以暂时被交换到**内存外的一个备份区**，随后可以被换回内存继续执行（中程调度）
b) 特点：打破进程运行的驻留性
c) 滚入，滚出——交换由于基于优先级的算法而不同，低优先级的进程被换出，这样高优先级的进程可以被装入和执行
d) 交换时间的**主要部分是转移时间**，总的转移时间直接同交换的内存的数量成比例

🧠 地址绑定三种策略：

策略类型	地址绑定时机	是否可以换入不同位置？
编译时绑定	编译时确定物理地址	❌ 不能换地址（必须原地）
装入时绑定	装入内存时确定地址	✅ 可以重新分配地址
执行时绑定	运行时由MMU映射地址	✅ 完全自由

e) 正在执行 IO 操作的进程不能被 swapping——IO 的物理地址已经被链接，一旦换出换入后进程的物理地址因为绑定策略发生变化，可能导致 IO 数据错误写入

四、存储管理方式

主存一般被分为两部分：低地址空间存放操作系统进程，高地址空间容纳用户程序

a) 连续内存分配

i. 单一连续分配：用户区只能容纳一道作业——适合单道程序，会产生内碎片

ii. 多分区分配

1. 固定分区大小分配：在作业被装入前，内存就被划分为若干个固定大小的连续分区

a) 在系统运行期间，分区的大小不可改变（但是在划分时分区大小可以相同也可以不同）——静态分区

b) 系统有一张**分区说明表**，每个表目说明一个分区的大小、起始地址和是否已分配的使用标志

c) 缺点：易产生内碎片；分区个数固定，限制并发执行的进程个数

2. 可变分区大小分配：寻找某个空闲分区，其大小需大于或等于程序的要求。若是大于要求，则将该分区分割成两个分区，其中一个分区为要求的大小并标记为“占用”，而另一个分区为余下部分并标记为“空闲”。分区的先后次序通常是从内存低端到高端。——动态分区

a) 寻找满足条件分区的方法：

i. 首次适应

ii. 最佳适应：适应且最小的

iii. 最差适应：适应且最大的

b) 分页(Paging)——非连续内存分配：

i. 分页是解决外碎片的方法

ii. 每个进程都有一个自己的页表（可以存储页号是否有效）

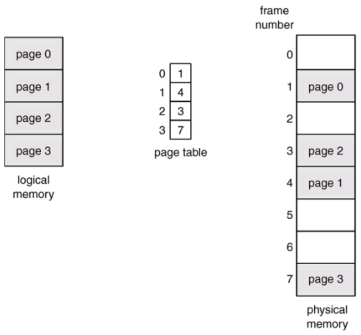
iii. 把物理内存划分为大小固定的块，叫做帧(frame)

把逻辑内存也划分为固定大小的块，叫做页(page)

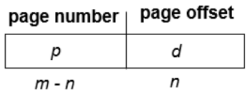
在分页系统中，页大小和帧大小是严格相等的

通过建立页和帧的映射关系，建立一个页表，用于记录逻辑地址与物理地址的转换关系

iv. 由 CPU 生成的虚拟地址通常由两部分构成：页号(page number)和页内偏移(page offset)。页号是页表的索引，用于在页表中定位到内存帧号（在内存的基地址），基地址+页内偏移就是物理内存地址



由CPU生成的每个地址分成两部分：页码page number（p）和偏移page offset（d）。页码作为页表的索引，页表包含每页所在物理内存的基地址，基地址加偏移就形成了物理内存地址。



v. 求解物理地址过程：已知页大小、虚拟地址
   1. 根据页大小划分虚拟地址（第 n 位为页内偏移，高 m-n 位为页号）
   2. 根据页号，查页表，确定帧号
   3. 帧号*页大小=基地址
   4. 物理地址=基地址+页内偏移

vi. 快表(TLB):
   1. 正常的页表存储在主存中，每次寻址需要两次访存，为了提升寻址速度，采用一个 Cache 存储部分页表项，每次访存前先查找快表，若页号在 TLB 中，直接取出对应的帧号，若没有命中，再访存，并利用局部性原理更新 TLB

vii. 分页的保护:
   1. 地址越界保护
   2. 通过页表的访问控制信息对内存信息提供保护（页表的每个表项中包含有效和无效两种状态：有效表示相关的页在进程的逻辑地址空间，是一个合法的页；无效表示页不在进程的逻辑地址空间）

viii. 页表结构
   1. 多级页表
   2. 哈希页表
   3. 反转页表：不再是给每个进程创建一个页表，而是为整个物理内存建造一张表，内存中的一块占用表中的一项，每个表项包含进程的逻辑页号和进程表示

ix. 总结：逻辑地址和物理地址的转换：
   1. 逻辑地址%页大小=页号
   2. 已知页号，查页表，找到对应的物理帧号
   3. 逻辑地址 mod 页大小=页内偏移
   4. 物理帧号*页大小+页内偏移=物理地址

c) 分段：一个程序是一些短的集合，一个段是一个逻辑单位

i. 核心思想：分段是将一个程序根据“逻辑意义”划分成多个段，每个段的大小不固定，由程序结构决定

分段与分页的比较

	分页	分段
目的	为了提高内存的利用率；	为了能更好地满足用户的需要。
单位划分	**页是信息的物理单位，页的大小是固定的，而且由系统确定。**	段是**信息的逻辑单位**,它含有一组意义相对完整的信息。段的长度是不固定的，取决于用户所编写的程序，并由编译程序来划分。
作业地址空间	单一的线性地址空间	二维的，标识一个地址需给出段名和段内地址。
内存分配	以页为单位离散分配，无外碎片，所以也无紧缩问题	以段为单位离散分配，类同可变分区，会产生许多分散的小自由分区——外碎片，造成主存利用率低，需采用紧缩解决碎片问题，但紧缩需花费时间

ii. 解决分段产生的外碎片：紧缩

iii. 段表包含：段的基地址+段长

第十章 虚拟内存

一、背景知识

a) 如果进程大于内存的容量或内存中同时运行多个进程：**从逻辑上扩充内存容量**

b) 常规存储器的特征：

i. 一次性：作业在运行前需要一次性全部装入内存

ii. 驻留性：作业装入内存后，便一直驻留在内存中，直到作业结束

c) 局部性原理：在一段时间内，程序的执行仅局限于某个部分；相应地，它所访问的存储空间也局限于某个区域内。

i. 时间局部性

ii. 空间局部性

d) 虚拟内存：允许进程部分装入内存就可以执行的技术——允许页面的换入换出（逻辑地址空间比物理地址空间大）

e) 实现虚拟内存的手段：请求页式——在**分页的基础上，增加了请求调页和页面置换功能后形成的页式虚拟存储系统**

f) 虚拟内存的特点：

i. **离散性**:在内存分配时采用离散的分配方式，是虚拟存储器的最基本的特征。

ii. **多次性**:一个作业被分成多次调入内存运行，即在作业运行时没有必要将其全部装入，只须将当前要运行的那部分程序和数据装入内存即可。是虚拟存储器最重要的特征。

iii. **对换性**:作业运行过程中信息在内存和外存的对换区之间换进、换出。

iv. **虚拟性**:从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。

二、请求调页(Demand Paging)

a) 预调页：主动的页面调入策略，即把那些预计很快会被访问的程序或数据所在的页面，预先调入内存。（主要用于进程的首次调入）

b) 请求调页：当进程在运行中发生缺页（产生缺页中断）时，由系统将缺页调入内存。

c) 页表机制（页表项）：

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

> **状态位（存在位P）**：用于指示该页是否已调入内存，供程序访问时参考。
>
> **访问字段A**：用于记录本页在一段时间内被访问的次数，或最近已有多长时间未被访问，提供给置换算法选择换出页面时参考。
>
> **修改位M**：表示该页在调入内存后是否被修改过。由于内存中的每一页都在外存上保留一份副本，因此，若未被修改，在置换该页时就不需将该页写回到外存上，以减少系统的开销和启动磁盘的次数；若已被修改，则必须将该页重写到外存上，以保证外存中所保留的始终是最新副本。--->提高性能
>
> **外存地址**：用于指出该页在外存上的地址，通常是物理块号，供调入该页时使用。

d) 缺页中断：请求分页系统中，每当所要访问的页面不在内存时，便要产生一缺页中断，请求 OS 将所缺页调入内存。

i. 与一般中断的**主要区别**

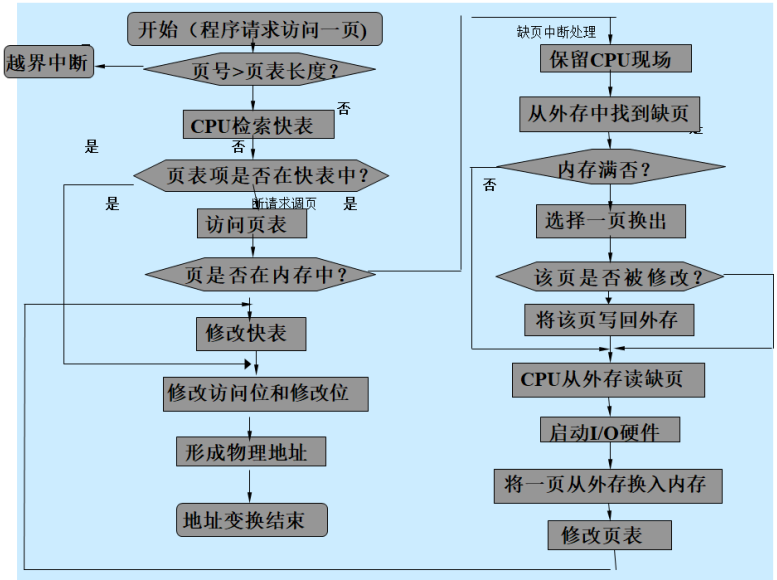
1. 缺页中断在**指令执行期间产生和处理**中断信号，而一般中断在一条指令执行完后检查和处理中断信号。
2. 缺页中断返回到该指令的开始重新执行该指令，而一般中断返回到该指令的下一条指令执行

e) 总结：在请求分页中，由 CPU 给出一个虚拟地址，计算出页号后查找页表，找到对应的物理帧号，如果该页表项无效，则说明当前页不在内存中，触发缺页中断，到外存（磁盘）上找到对应的物理块，采用页面置换算法将其加载到内存中，更新当前进程的页表（将内存帧号填入到页表中），重新开始被中断的指令

三、页面置换

a) 如果没有空闲帧，需要两个页面传输，一个换出，一个换入

b) 置换实现了逻辑内存和物理内存的划分—在一个较小的物理内存基础之上可以提供一个大的虚拟内存



c) 页面置换算法：
   i. 最佳算法：被置换的页**将是之后最长时间**不被使用的页——实际过程中无法得到以后会引用的所有页面的信息，因此无法实现
   ii. 先进先出(FIFO): FIFO 算法可能会产生 **Belady 异常**，就是**更多的页框反而会产生更多的缺页**
   iii. 最近最久未使用(LRU)

四、页帧的分配：
a) 如何给进程分配一定的空闲内存？
   i. 分配策略：
      1. 平均分配
      2. 按比例分配：根据每个进程的大小来分配
      3. 优先分配：根据优先级分配
b) 全局替换和局部替换：
   i. 全局替换：进程在所有的页中选择一个替换页面；一个进程可以从另一个进程中获得页面
   ii. 局部替换：每个进程只从属于它自己的页中选择

五、颠簸（抖动）
a) 定义：置换出去的页立刻又需要置换进来，因此，会不断的产生缺页（刚被换出的页很快又被访问，需重新调入，导致系统频繁地交换页面，以致大部分 CPU 时间花费在完成页面置换的工作上。这样会造成 **CPU 利用率低下**）
b) 全局置换会造成颠簸，局部置换能限制系统颠簸，但是也会增加进程的有效访问时间，因此应该给进程足够的帧以防止颠簸

第十一章 文件系统

一、基本概念
a) 内存具有易失性，需要文件系统管理信息
b) 文件系统是操作系统的重要组成部分，负责信息的组织、存储和访问
c) 文件系统的功能就是**提供高效、快速和方便的信息存储和访问功能。**
d) **文件系统是**操作系统中以文件方式管理计算机软件资源的软件和被管理的文件和数据结构（如目录和索引表等）的集合。
e) **从系统角度来看，文件系统是对文件存储器的存储空间进行组织、分配和回收，负责文件的存储、检索、共享和保护。**
   从用户角度来看，文件系统主要是实现“按名取存”，文件系统的用户只要知道所需文件的文件名，就可存取文件中的信息，而无需知道这些文件究竟存放在什么地方。
f) 文件系统的作用：
   i. 提供对文件的各种操作，实现按名存取
   ii. 提供合适的访问方式

iii. 提供目录管理和操作
iv. 实现文件的共享、保护
v. 统一管理文件的存储空间，实现存储空间的分配和回收
vi. 实现逻辑文件与物理文件间的转换

g) 文件：是记录在外存上的具有名字的相关信息的集合
h) 文件属性：文件名、类型、大小、位置、保护（读取、改写权限）、时间.....
属性信息放在**目录**中存储在硬盘里
i) 文件系统的结构：
i. 文件系统按层组织（具有明显的分层结构）：
1. 用户层（应用层）：用户与文件系统交互的接口
2. 逻辑文件系统：管理文件的元数据（文件名、路径、用户权限等）
3. 文件组织模块：将文件分成多个逻辑块并建立映射关系
4. 基本文件系统：管理磁盘上的物理块，将逻辑块号映射为实际磁盘地址
5. IO 控制器：完成对磁盘的读写

二、文件结构与存储设备

a) 文件结构
用户按逻辑结构使用文件，文件系统按物理结构管理文件。因此，当用户请求读写文件时，文件系统必须实现文件的逻辑结构与物理结构之间的转换。
i. 文件有两种形式的结构：
1. 逻辑结构：用户对文件的组织结构
文件从逻辑结构上分为两类：
a) 无结构的流式文件：对文件内信息不再划分单位，依次是一串字符流构成的文件
b) 有结构的记录式文件：用户把文件内的信息按照逻辑上独立的含义划分信息单位，每个单位称为一个逻辑记录(简称记录)
记录文件分为顺序、索引、索引顺序三种
i. 顺序结构文件：所有记录按键值的约定次序组织，常用于批量记录的读取
ii. 索引文件：对主文件的记录按照需要的数据项建索引表（为每个记录设置一个表项），常用于随机访问
iii. 索引顺序文件：将顺序文件的所有记录分组，再为顺序文件建立索引表，表中记录每组的第一个记录
2. 物理结构：文件在磁盘上的存储结构
文件的存储设备（主要是磁盘、光盘）常常将物理空间划分为若干个大小相等的块，以块为单位进行信息的存储、传输
a) 顺序分配：每个文件占用一个连续的磁盘块的集合——会产生外碎片
b) 链接分配：文件的存储设备（主要是磁盘、光盘）常常将物理空间划分为若干个大小相等的块——不支持随机访问，不会产生外碎片
i. FAT：（文件分配表）是一种典型的链式文件分配结构，它通过一张集中管理的表（在内存中）记录每个文件的磁盘簇链，使得文件可以存储在非连续的磁盘区域中，避免外部碎片；每个文件的目录项中保存起始簇号，通过 FAT 表即可遍历完整文

件，指针不在数据块中而集中于表中，是对传统链式存储的优化

c) 索引结构：系统为每个文件建立一个索引表，将不连续的块号存储在索引表中（一个索引表就是磁盘块地址数组,其中第 i 个条目指向文件的第 i 块）

三、目录结构

a) 目录：包含所有文件信息的节点的集合（目录结构和文件都在磁盘上）——FCB 的有序集，是实现按名存取的重要手段

b) 目录中的信息：名称、类型、地址、当前长度、最大长度、最后访问时间、数据最后更新时间、所有者 ID、保护信息

c) 目录以文件的形式组织，存放在磁盘上

d) 对目录的操作：

i. 建立、寻找、删除一个文件

ii. 列出目录的列表

iii. 重命名文件

iv. 遍历文件系统

e) 基于索引的文件系统（如 ext3/ext4）中，每个文件都有一个自己的 inode（索引节点）；目录文件本质上是一个包含多个“目录项”的文件，每个目录项记录了一个“文件名 → inode 号”的映射。系统通过目录项中的 inode 号定位到该文件的 inode，进而访问文件内容。

第十二章 二级存储

一、磁盘结构：

a) 柱面：各盘面所有的读写头同时移动，并定位在同样的垂直位置的磁道上，这些磁道形成了一个柱面。

b) 磁头：磁盘的全部有效盘面从上到下依次编号

c) 扇区：将各盘面分割成若干大小相等的扇区

d) 数据首先都映射到一个磁道，其余的数据映射到同一柱面的其他磁道，然后按照从外向里的顺序映射到其余的柱面。

e) 盘块的物理地址由柱面号、磁头号、扇区号三部分组成。

设L，M，N分别为盘组的柱面数、盘面数、扇区数，B表示块号，则第i柱面、j磁头、k扇区所对应的块号B为：

B=（i×M ×N）+（j ×N）+k，其中i=0，…,L-1; j=0,…,M-1; k=0, …, N-1

根据块号B也可以确定该块在磁盘上的物理位置.

> **柱面号:** i=int (B, M ×N)
>
> **磁头号：** j=int (mod(B, M ×N), N)
>
> **扇区号：** k=mod (mod (B, M ×N), N)

f) 提高磁盘 IO 的速度：

i. 设置磁盘高速缓冲区

ii. 采用好的磁盘调度算法

二、磁盘调度算法：

a) 磁盘访问的相关时间：

i. 寻道时间：是指磁头在径向方向上移动的时间，即在多个柱面之间移动的时间（一般不考虑在多个磁头之间的选择时间）

ii. 旋转延迟：指定扇区移动到磁头下所经历的时间（与磁盘的转速有关）
平均旋转延迟：Tr=1/2r(r 为磁盘转速)——平均旋转延迟是总旋转延迟的一半

iii. 传输时间：把数据从磁盘读出或者像磁盘写入的时间：Tt=b/rN(b 为所读/些的字节数；r 为磁盘的旋转速度；N 为一条磁道上的字节数)

总访问时间

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

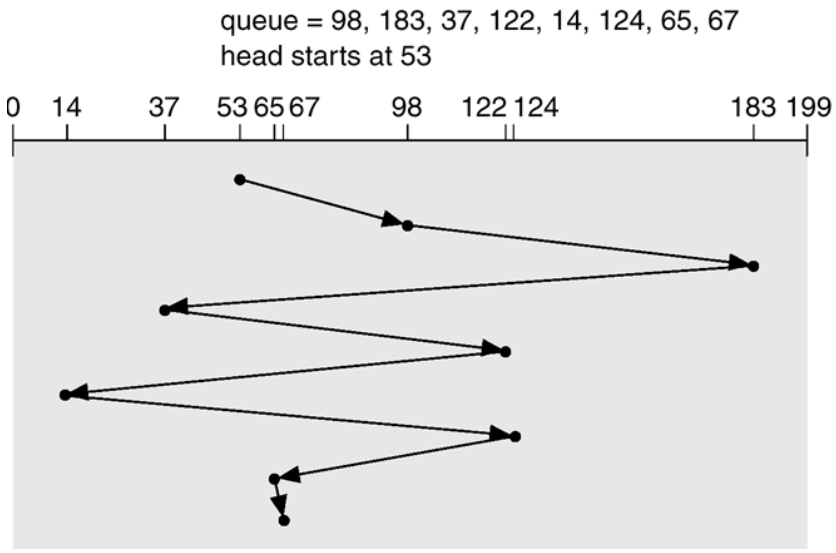
寻道时间和旋转延迟时间，基本都与所读／写数据的多少无关，而且它通常是占据了访问时间的大头。

目前随着磁盘传输速率的不断提高，数据传输时间所占的比例更低。可见，适当地集中数据(不要太零散)传输，将有利于提高传输效率。

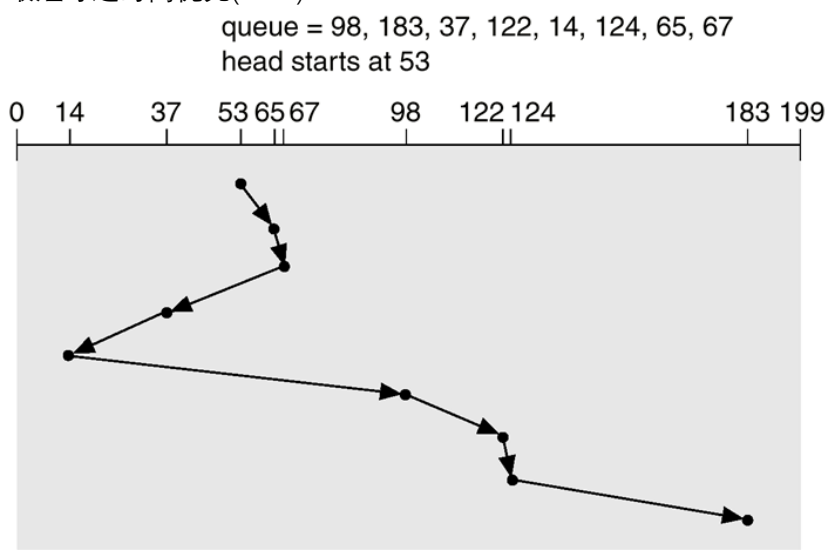
iv.

b) 磁盘调度算法：

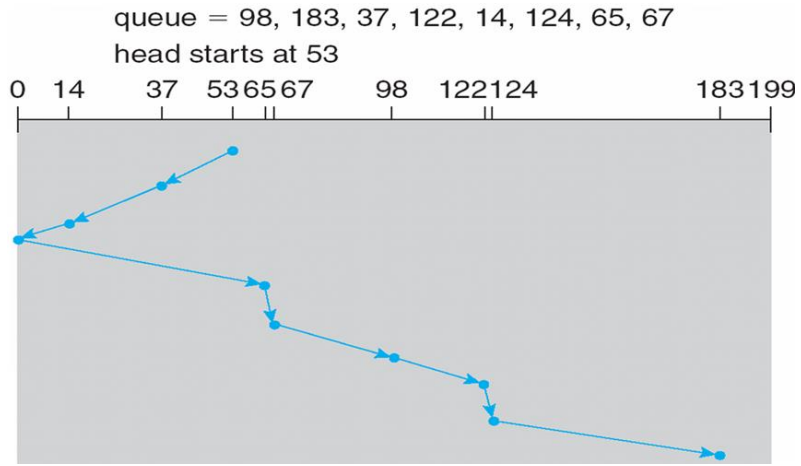
i. 先来先服务(FCFS)



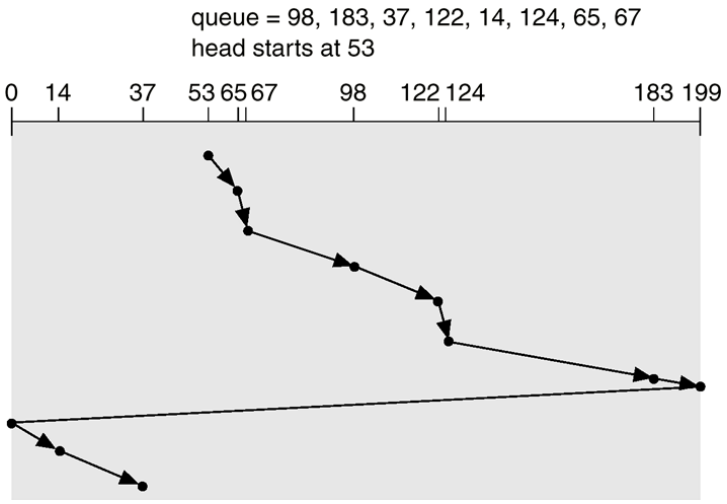
ii. 最短寻道时间优先(SSTF)：



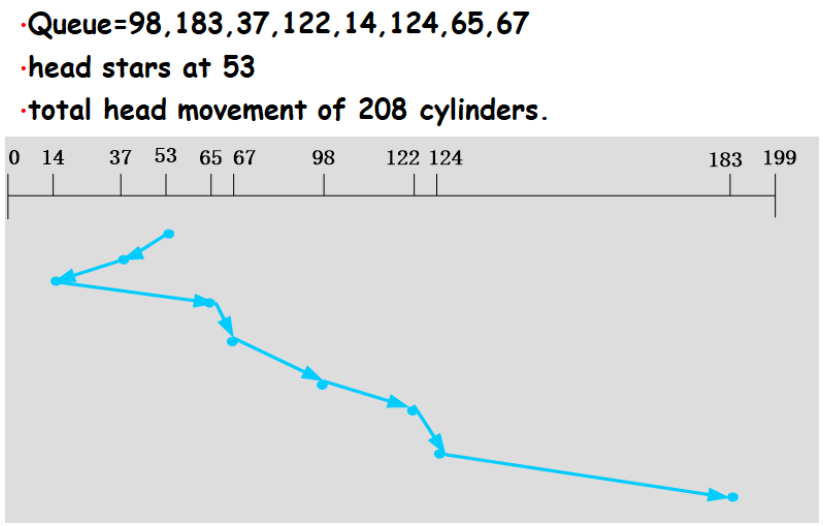
iii. Scan 算法(扫描算法)：算法所选择的方向一侧移动，直到再无更外的磁道需要访问时，才更换方向（注意：一定要走到头）



iv. C-scan（循环扫描算法）：磁头从磁盘的一段向另一端移动，沿途响应请求。当它到了另一端，就立即回到磁盘的开始处，在返回的途中不响应任何请求。



v. LOOK 算法：磁臂在每个方向上仅仅移动到最远的请求位置，然后立即反向移动，而不需要移动到磁盘的一端（注意：不一定要走到头，走到请求的最远端即可）



第十三章　IO 设备

一、IO 系统的组成

1. IO 设备：
   a) 按速率分类：
      i. 低速设备：键盘、鼠标
      ii. 中速设备：打印机
      iii. 高速设备：磁盘
   b) 按信息交换的单位分类：
      i. 块设备：用于存储信息
      ii. 字符设备：用于数据的输入和输出
   c) 按设备的共享属性分类：
      i. 独占设备：在一段时间内只能有一个进程使用的设备（一般为低速设备）
      ii. 共享设备：一段时间内可以有多个进程共同使用（如硬盘）
      iii. 虚拟设备：通过虚拟技术把一台独占设备变换成若干台逻辑设备，可供多个用户使用
2. 设备控制器：
   a) 设备并不是直接与 CPU 进行通信，而是与设备控制器通信，某些设备有内置的控制器
   b) 每个 I/O 设备通过设备控制器与计算机的数据总线和地址总线相连接。
3. IO 通道
   a) **定义**：通道是独立于 **CPU 的专门负责数据 I/O 传输工作的处理机**，对外部设备实现统一管理，**代替 CPU 对 I/O 操作进行控制**，从而使 I/O 操作可与 CPU 并行操作。 通道可以执行通道程序.
   b) 通过 IO 通道传输的数据最终存储在内存中的某个缓冲区内

二、IO 控制方式：

1. 程序 IO（轮询）
2. 中断驱动：每传输完一个字（或者字节）的数据，就产生一次中断
3. DMA（直接存储器访问）：以块为单位进行数据传输，数据直接从 IO 送入内存

三、缓冲池

1. 引入缓冲的主要原因
   a) 缓和 CPU 与 I/O 设备间速度不匹配的矛盾
   b) 减少对 CPU 的中断频率，放宽对中断响应时间的限制
   c) 提高 CPU 和 I/O 设备之间的并行性
2. 工作过程
   a) 输入过程：
      i. [键盘 / 文件] → (收容输入缓冲区) → [输入缓冲区] → (提取输入缓冲区) → 程序使用
   b) 输出过程：
      i. 程序输出 → (收容输出缓冲区) → [输出缓冲区] → (提取输出缓冲区) → [终端 / 文件]

四、IO 软件

1. IO 软件按照分层的思想构造
   a) 较低层的软件要使较高层的软件独立于硬件
   b) 较高层的软件要向用户提供一个良好的界面
2. IO 软件的四个层次：
   a) 用户空间的 IO 软件
   b) 与设备无关的 IO 软件（设备独立软件）
   c) 设备驱动程序
   d) 中断处理程序

五、设备分配：

设备分配

在多道程序环境下，系统中的设备**不允许**用户自行使用，必须由系统分配。

为了实现设备分配，必须在系统中设置相应的数据结构。

在进行设备分配时所需的数据结构有：

> 设备控制表DCT
>
> > 系统为每一个设备都配置了一张设备控制表，用于记录本设备的情况
>
> 控制器控制表COCT
>
> 通道控制表CHCT
>
> 系统设备表 SDT
>
> > 整个系统有一张系统设备表