绪论

操作系统是控制和管理计算机系统内各种硬件和软件资源、有效地组织多道程序运行的系统软件，是用户和计算机之间的接口。

定义，功能

操作系统是介于计算机和应用软件之间的一个软件系统，操作系统的上层和下层都有其他对象存在。

操作系统的目标：方便性、有效性（提高系统资源利用率、提高系统吞吐量）、可扩充性、开放性

操作系统的作用**

作为用户和计算机之间的接口，OS处于用户与计算机硬件系统之间，通过一下方式交互：

1. 命令方式
2. 脱机命令接口（Windows）= 批处理命令
3. 图形、窗口方式
4. 系统调用方式（直接给程序使用）

实现了对计算机资源的有效管理：处理机管理功能；文件管理；IO设备管理；内存管理。

特征

- 并发：区别并行和并发；引入进程
- 异步
- 共享：互斥共享方式；同时访问方式。并发和共享是多用户（多任务）OS的两个最基本的特征，互为存在的条件
- 虚拟：时分复用技术（虚拟处理机技术、虚拟设备技术、虚拟信道技术）；空分复用技术（虚拟存储器）

并发性：
并发性是两个或多个事件在同一时间间隔内发生（宏观上同时发生，微观上交替发生）；
并行性则是在同一时间间隔内发生（多处理器的支持）。

共享：
系统中的资源可供内存中的多个并发执行的进程（线程）共同使用。

1. **互斥共享方式**：为使输出结果不混淆，规定一段时间内只允许一个进程（线程）访问该资源。当当下进程访问完并释放资源后，才允许另一进程对该资源访问。在一段时间内只允许一个进程访问的资源称为临界资源或独占资源。
2. **同时访问方式**：以并发执行为条件允许一段时间内由多个进程"同时"对资源进行访问（宏观上同时，微观上交替对资源访问）。如果系统不能对资源共享实施有效管理，无法并

虚拟化：

1. **虚拟性**：把一个物理实体变为若干个逻辑上的对应物（虚拟）。
2. **虚拟存储技术**：物理存储器变为虚拟存储器，从逻辑上扩充存储器容量。

3. **虚拟设备技术**：一台物理IO设备虚拟为多台逻辑上的IO设备，并允许每个用户占用一台逻辑上的IO设备。
4. 把一条物理信道寻味多条逻辑信道（虚信道）

异步性：
多道程序环境下，允许多个进程并发执行。

发展与分类

发展：未配置操作系统的计算机系统 -> 单道批处理系统 -> 多道批处理系统 -> 分时系统 -> 实时系统

人工操作方式：
用户独占全机，CPU要等待人工操作。
后来引入脱机输入/输出方式（用外围机+磁盘实现），并有外围处理机负责控制作业的输入输出。

单道批处理系统：
单个处理器、后备作业队列、IO设备、监管程序 共同构成系统。
但内存中仅有一道程序运行，CPU有大量时间在空闲等待IO完成。

多道批处理系统：
多道性、无序性、调度性。
资源利用率高，系统吞吐量大；但平均周转时间长，无交互能力。

分时系统：
人机交互、共享主机、便于用户上机。多个用户同时输入命令，系统也应能全部地及时接收并处理。
用户请求可以被及时响应，解决人机交互问题，允许多个用户同时使用一台计算机，用户对计算机操作相互独立；
但不能优先处理紧急任务，对各个用户/作业都是完全公平的，循环地为每个用户/作业服务一个时间片，不区分任务的紧急性。

实时系统：
及时性、可靠性。
系统能及时（或即时）响应外部事件的请求，在规定时间内完成对该事件的处理，并控制所有史诗任务协调一致地运行。
可分类为：

1. 硬实时任务：系统必须满足任务对截止时间的要求，否则会出现难以预测的结果。
2. 软实时任务：对截止时间不太严格，偶尔错过也不会对系统产生太大影响影响。

操作系统的运行环境

运行机制、中断、系统调用。

运行机制

- CPU状态
    - 用户态（常态、目态）
        - 执行非特权指令
    - 内核态（核心态、特态、管态）：执行全部指令
        - 当一个任务执行系统调用而进入内核代码中执行时，进程处于内核态，执行管补指令

    - 特权指令和非特权指令：
        - 特权指令只允许操作系统内核部分使用，不允许用户直接使用（包括IO指令、置终端屏蔽指令、清内存、见存储保护、设置时钟指令）。处于内核态运行的是内核程序，可以执行特权指令
        - 非特权指令所有程序均可使用。处于用户态运行的是应用程序，只能执行非特权指令
    - 内核态与用户态的转换
        1. 系统调用：用户态进程主动要求切换为内核态的方式之一
        2. 异常：发生异常时会触发有当前运行进程切换到处理此异常的内核相关程序中
        3. 外围设备中断：外围设备完成用户请求后向CPU发出相应中断信号，此时CPU会暂停执行下一条即将执行的指令转去执行与中断信号对应的程序
- CPU执行的程序
    - 操作系统内核程序：操作系统的核心，最接近硬件的部分；OS的功能未必都在内核
    - 应用程序
        - 用户态

中断和异常

*中断*是计算机执行程序过程中出现异常情况或特殊请求，计算机停止运行现行程序转而运行这些*异常情况*或特殊请求的处理，处理结束后再返回现行程序。分别有：
内中断（异常、例外）：【三种类型】陷阱、陷入(trap)，故障(fault)，终止(abort)。【九种异常】缺页，除数为零，越界错误，存储保护错，非法指令，特权指令，运行超时，等待超时，算术运算错。 **外中断（中断）**：时钟中断：IO中断请求。

*故障*是引起故障的指令在执行过程中CPU检测到一类与指令执行相关的意外事件。有些可恢复，有些不能。
溢出和非法操作码则无法恢复；
除数为0的情况则根据定点除法指令或浮点出发指令有着不同的处理方式。

*陷阱*是预先安排的"异常"时间，如同预先设定的"陷阱"。执行到陷阱指令时，CPU会调出特定程序进行相应处理。
陷阱提供了用户程序和内核之间一个像过程一样的接口（系统调用）

*终止*是在指令执行过程中发生了严重错误使程序无法继续执行，被迫终止运行。

系统调用

提供了操作系统提供的有效服务界面，一种可供应用程序调用的特殊函数，通过系统调用请求获得操作系统的内核服务。

调用者不需要知道调用或执行过程中做了什么，只需要遵循API并了解执行系统调用后系统做了什么。

系统调用类型：
与共享资源相关操作（内存分配、IO操作、进程管理、文件管理、设备管理、信息维护、通信等）均通过系统调用的方式向操作系统内核提出服务请求，由操作系统内核代为完成。其过程为：
`应用程序 -调用-> API -存入系统调用号并触发中断[int 0x80]进入-> 中断处理函数 -根据调用号调用-> 系统调用 -返回-> 中断处理信号 -返回-> API -返回-> 应用程序`

`传递系统调用参数 --> 执行陷入指令（用户态） --> 执行相应的内请求和程序处理系统调用（核心态） --> 返回应用程序`

操作系统的设计

操作系统的功能：内存管理，处理机管理，进程管理，文件管理，IO管理

- 传统操作系统结构
  - 无结构操作系统
  - 模块化OS
  - 分层式结构OS
- 微内核OS结构（现代操作系统结构）
  - 客户/服务器模式
  - 面对对象的程序设计

微内核OS结构
(非内核) OS的图形界面
(内核) 进程管理；内存管理；文件管理；IO管理
(内核) 时钟管理 / 中断管理 / 原语
硬件（裸机）

原语是一种不可被"中断"的特殊程序

微内核技术是能实现OS核心功能的小型内核，运行在核心态，开机后常驻内存。
微内核OS分为两部分：一部分用于提供各种服务的一组服务器（进程），另一部分是内核。

- 内核功能：
  1. 进程（线程）管理
  2. 低级存储器管理（用户程序逻辑空间到内存空间的物理地址的变换）
  3. 中断和陷入管理（中断和陷入）
- 具体功能：
  1. IO指令、置终端屏蔽指令、清内存、建存储保护、设置时钟指令
  2. 中断、异常、陷入
  3. 进程（线程）管理
  4. 系统调用
  5. 用户内存地址转换（逻辑 --> 物理映射）

操作系统的启动

开机时运行一个初始化程序（引导程序bootstrapProgram，位于ROM或EEPROM，称为固件），引导程序定位操作系统内核并装入内存，接着操作系统开始执行第一个进程并等待事件发生。

1. bois引导：`开机 --> BIOS启动MBR --> MBR启动激活分区PBR --> 启动bootmgr --> 读取BCD --> 启动对应的系统`
2. UEFI引导：`开机 --> BIOS找到第一个硬盘 --> BIOS搜索BOOTx64.efi（计算机默认引导）或 bootmgrfw.efi（Windows默认引导） --> 读取BCD --> 启动BCD对应菜单的系统`

虚拟机

单个计算机硬件抽象为多个不同的逻辑实体。

补充内容

- 单处理机系统中，进程与进程是不可并行的。
- 多道程序系统由于需要调度多个程序，所以系统开销相对变大。
- 不可能在用户态发生的事件是进程切换。缺页、系统调用可能在用户态发生（但必须在内核态执行）
- 访存时缺页属于异常

进程管理

内核的一个主要功能，在内核态完成

进程的概念和特征

进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。

当前一操作（程序段）执行完成后才能执行后续操作，执行期间要从前往后顺序执行。但一旦引入并发程序后，程序将有间断性、失去封闭性、不可再现性。

进程的特征

1. **结构特征**：程序段、数据段、PCB 三个部分组成进程实体
    - 进程控制块PCB：OS根据PCB来对并发挥自i选哪个的进程进行控制和管理。其所包含的信息有：
        - 进程标识符：外部标识符PID，内部标识符(端口)
        - 处理机状态：通用寄存器，指令计数器，程序状态字PSW，用户栈指针
        - 进程调度信息：进程状态，进程优先级，进程调度所需的其它信息，事件
        - 进程控制信息：程序和数据的地址，资源清单，进程同步通信机制，链接指针
2. **动态性**：进程的最基本特征，由创建而产生、由调度而执行、由撤销而消亡。
3. **并发性**：多个进程实体同存于内存，且能在一段时间内同时运行。*并发*是多个进程轮替执行，*并行*是同时执行。
4. **独立性**：进程实体是一个能独立运行、独立分配资源、独立接受调度的基本单位。但凡为建立PCB的程序都不能作为一个独立单位参与运行。
5. **异步性**进程按各自独立的、不可预知的速度向前推进（进程实体按异步方式运行）。

进程控制块的组织方式

1. 链接方式：把进程按照链表的形式连接起来。
2. 索引方式：用索引表来存储进程控制块的位置信息。

进程和程序的关系

进程是程序在数据集上的一次执行；程序是构成进程的组成部分，一个程序可对应多个进程，一个进程可包含多个程序；集成度运行目标是还行所对应的程序；从静态看，进程由程序、数据、进程控制块PCB组成。

进程	**程序**
动态	静态
是程序在数据集上的依次执行，有创建、撤掉，存在是暂时的	是指令的有序集合，永远存在
有并发性	无并发性
进程可创建其它进程	程序不能形成新的程序
竞争计算机资源的基本单位	不参竞争计算机资源

进程的状态和切换

进程的五种基本状态

- 创建状态：正在创建，不能运行，未提交，等待系统完成创建进程的所有必要信息，并初始化PCB
- 就绪状态：具备运行条件，等待分配CPU
- 执行状态：占用CPU并运行
- 阻塞状态：因某件事发生导致暂时无法运行（例如等待IO操作）
- 终止状态：执行完最后语句，并通过系统调用exit()请求OS终止进程，并回收内存空间和PCB

五种状态的转换

- 创建：创建进程->
- 就绪：调度进程->
- 运行：->
    - 终止进程->终止
    - 等待IO操作/Buffer/执行P操作/申请内存/申请设备->阻塞
        - 等待事件完成->就绪
    - 时间片用完/CPU被剥夺/CPU被高优先进程抢占->就绪

进程控制

对系统中的所有进程实施有效管理，具有创建新进程、撤销已有进程、实现进程状态转换等功能。进程控制具有原子性，属于原语结构。
进程之间的组织关系是进程图。

1. 进程创建：进程创建原语，引起创建进程事件、进程的创建过程、创建完成
2. 进程终止：检索被终止进程的PCB，立即终止进程、调度标志置为真，子孙进程终止，进程拥有资源归还父进程或系统，被终止进程的PCB从所在队列或链表中移出
    1. 正常结束：一个用于表示进程已经运行完成的指示。
    2. 外界干预（进程相应外界请求而终止运行）：操作员或操作系统敢于、父进程请求、父进程终止
    3. 异常结束：出现错误或故障被迫进程终止
3. 进程阻塞：进程无法继续执行时调用阻塞原语block把自己阻塞，进程控制块由“执行”改为阻塞，PCB插入阻塞队列，调转读程序进行重新调度
    - 原因：请求系统服务，启动某种操作，新数据尚未到达，无新工作可做
4. 进程唤醒：被阻塞进程所期待事件出现，有关进程调用唤醒原语wakeup()
5. 进程挂起：进程被暂时调离出内存，条件允许时被操作系统再次调回内存，重新进入就绪状态。被挂起的程序由就绪改为静止就绪；阻塞改为禁止阻塞；正在被执行则专项调度程序重新调度。
    - 原因：用户请求，父进程请求，操作系统要求，负载调节要求，定时任务
6. 进程激活：父进程或用户进程请求激活指定进程，且内存已有足够空间，OS利用激活原语active()激活指定程序。静止就绪改为活动就绪；静止阻塞改为活动阻塞
7. 进程调度：

进程控制的原语

1. 进程创建原语：申请一个空白PCB，将调用者参数添入PCB，设置记账数据，设置新进程为“就绪”
2. 终止进程原语：终止已完成的进程，回收其所占资源
3. 阻塞原语：将进程从运行态变为阻塞态。进程被插入等待事件队列，同时修改PCB中相应表项
4. 唤醒原语：将进程从阻塞态变为就绪态。进程从阻塞队列移出，插入就绪队列等待调度，同时修改PCB相应表项

进程同步

两个及以上进程基于某个条件协调活动。

进程具有异步性。但进程与进程需要共享系统资源，将构成竞争关系，于是存在**制约关系**：

- 间接相互制约关系：互斥——竞争
- 直接相互制约关系：同步——协作

资源竞争会出现两个控制问题： **死锁deadlock**（一组进程获得了部分资源并想得到其它进程占有的资源，最终所有进程进入死锁）；
饥饿starvation（由于其它进程优先于它而被无限延期）

进程的互斥是解决进程间竞争关系的手段（若干个进程要使用同一资源时，最多允许一个进程去使用，其它进程等待。此过程要借助临界区）

临界区是进程访问临界资源的代码段（临界段）：进入区、临界区、退出区、剩余区。进入区和退出区是负责实现互斥的代码段。

进程同步的原则

1. 空闲让进：临界资源处于空闲状态，允许一个请求进入临界区的进程立即进入自己的临界区，以有效利用临界资源。
2. 忙则等待：临界区已有进程时，其它进程必须等待，保证临界资源的互斥访问。
3. 有限等待：保证进程在有限时间内能进入临界区，避免陷入死等
4. 让权等待：当进程不能进入临界区，应立即释放出立即，避免陷入忙等

1、2、3、必须满足，4、可以不被满足

进程同步的软件实现

单标志法、爽标志先检查、双标志后检查、Petersor算法

单标志法：
设置一个公共整型变量turn，代表是否允许进入临界区的进程编号。
但若某一进程不想进入临界区且turn==True，则其它进程也无法进入临界区，出现临界区资源空闲状态，违背*空闲让进*原则。

双标志先检查：
设置一个flag表示自己的访问状态，进程访问临界区前检查临界资源是否被占用，并设置自己的flag。这样不用交替进入，可以连续使用。
但两个进程会容易都进入临界区，违背*忙则等待*原则。

双标之后检查：
进程访问前先设置自己的flag，再检查其它进程的flag。两个进程都会表明自己要进入临界区的意愿。
但容易让两个进程都处于等待状态，不满足*有限等待*，导致饥饿。

Peterosn算法：
结合单标志法和双标志后检查，利用flag解决互斥访问，利用turn解决饥饿。
防止进程为了进入临界区而出现无限等待，再设置变量turn。进程设置flag后再设置turn标志。

在while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。
但不遵循*让权等待*，会发生忙等。

进程同步的硬件实现方法

关中断，利用Test-and-Set指令实现互斥，利用swap指令实现进程互斥

关中断屏蔽方法：
利用开关中断指令实现。整个指令执行过程不允许进程切换，不可能存在两个进程同时都在临界区。简单高效。
但不适合多处理机，只适合操作系统内核进程；让用户随意使用会影响整个CPU调度。

TestAndSet指令：
称TSL指令，配合互斥锁，确保只有一个进程在临界区。如果锁可用，进程继续执行并将锁设置为占用状态，其它进程会被阻塞，直到锁可用。
实现简单，不用严格检查逻辑漏洞，适用于多处理机。
但不满足让权等待，暂时无法进入临界区的进程会占用CPU并循环执行，循环指令TSL指令，导致忙等。

swap指令：
逻辑上和TSL指令无太大区别。临界区被占用，lock为true，交换后依然进入while循环；临界区空闲，old为false，跳出循环，进程进入临界区。
优缺点和TSL类似。

进程同步机制（信号量机制）

整型信号量、记录型信号量。

用户可以用操作系统的一对原语对信号量进行操作，便于实现进程同步、互斥。信号量是一个可以表示系统中某种资源数量的变量。

整型信号量

由Dijkstra把整型信号量定义为一个整型量，除初始化外，金童两个标准原子操作wait(S)和signal(S)访问（分别称为P、V操作）。

```
wait(S): while S≤0 do no-op
   S:=S-1
signal(S): S:=S+1
```

只要信号量S≤0，则会不断地测试。因此不遵守让权等待，使进程处于忙等。

记录型信号量

需要用于代表资源数目的整型变量value、进程链表L（链接所有等待进程）。

```
typedef struct{
   int value;  //剩余资源数
```

```
    struct process *L;    //等待资源队列
}semaphore;
```

```
procedure wait(S)
   var S: semaphore;
   begin
      S.value: = S.value-1;
      if S.value < 0 then block(S,L)
   end
procedure signal(S)
   var S: semaphore;
   begin
      S.value: =S.value+1;
      if S.value ≤ 0 then wakeup(S,L);
   end
```

S.value的初始值取决于系统汇总有多少可用资源数量。
当请求一个单位的资源时：`S.value: =S.value-1`，当S.value<0时，表示资源分配完毕，调用block自我阻塞，并插入到信号量链表S.L，遵循让权等待原则。
如果S.value初值为1，表示只允许一个进程访问临界资源，此时信号量转为互斥信号量。 signal操作则表示执行进程释放一个单位资源，若S.value+1后仍是S.value≤0，表示信号量链表中仍有等待该资源的集成被阻塞，古调用wakeup将表中的第一个等待进程唤醒。

管程机制

管程是代表共享资源的数据结构、以及由对该共享数据结构实时操作的一组过程锁组成的资源管理程序，共同构成了一个操作系统的资源管理模块。

由四部分组成：管程名称，局部于管程内部的共享数据结构说明，对该数据结构进行操作的一组过程，对局部于管程内部的共享数据设置初始值的语句。

管程的特性：模块化，抽象数据类型，信息掩蔽。

进程和管程对比：进程通过调用管程中的过程对共享数据结构进行实时操作，管程不能像进程那样并发执行（不能与其调用者并发），管程是操作系统中的一个供进程调用的资源管理模块（不具有动态性）。

管程相当于围墙。局部于管程内部的数据结构仅能被局部于管程内部的过程所访问。管程外与管程内的无法相互访问。所有进程要访问临界资源时，必须经过管程，且每次只允许一个进程进入管程，实现进程互斥。

一个进程被阻塞或挂起的条件/原因可能很多，因此管程中设置多个**条件变量condition**，每个条件变量保存一个链表用于纪录因该条件变量而阻塞的所有进程，**用x.wait、x.signal操作。**
当进程因x条件被阻塞/挂起，x.wait把此进程放到一个因x原因引起而阻塞的队列上。
当x条件变化，x.signal让x阻塞队列上的一个进程重新启动一个被阻塞的进程（如果没有被阻塞的过程，此操作不产生任何后果）。此signal不同于信号量机制的signal。

锁机制

锁让某一时间点只有一个线程进入临界区代码，保证数据一致性。锁的本质是内存中一个用于表示不同状态的整型数。
加锁时，判断锁是否空间，若空闲则修改为加锁状态并返回成功，若已上锁则返回失败，解锁时则修改为空闲状态。

锁的分类

1. **重量级锁**：获得不了锁时立即进入阻塞。
2. **自适应自旋锁**：根据线程最近获取锁的状态调整循环次数的自旋锁，称为自适应自旋锁。
3. **轻量级锁**：引入一个变量标记使用情况。若未在使用，则当进入此方法时采用CAS机制把状态编辑为已有人在执行，退出时改为无人执行。
4. **偏向锁**：偏向锁认为很少有两个线程执行一个方法，于是没必要加锁。
5. **悲观锁和乐观锁**：
    - 悲观锁认为不事先加锁总会出事。重量级锁、自旋锁、自适应自旋锁属于悲观锁。
    - 乐观锁认为不加锁没事，出现冲突了再解决（例如CAS机制）。轻量级锁属于乐观锁。

互斥锁

1. **LockOne类**：引入一个标志数组flag用于标记进入临界区的意愿。但当两个线程都表达想进入临界区的意愿，则会死锁。
2. **LockTwo类**：当两个进程都表达想进入临界区的意愿时，牺牲较晚对victim赋值的线程，以避免死锁。但只有一个进程时，会牺牲自己。
3. **Peterson锁**：LockOne和LockTwo的结合。
4. **Barkey锁**：n线程锁算法。进程进入临界区前flag表达意愿，并得到一个序号，序号最小的线程才能进入临界区。

进程通信

进程之间的信息交换。进程之间存在相互制约的关系（间接，直接）。

进程间通信机制IPC：进程之间交换数据必须通过内核，把进程1的数据从用户控件拷贝到内核缓冲区，进程2再从缓冲区读取。

低级通信：信号量机制，管程。
此通信方式效率低（生产者每次只能向缓冲池投放一个消息，消费者每次只能从缓冲区获取一个消息），通信对用户不透明

高级通信方式：共享存储器系统，消息传递系统，管道通信。

共享内存

多个进程直接读取同一块内存空间，最快的可用IPC形式。在内核专门留出一块内存区，由需要访问的进程将其映射到自己的私有地址空间供进程直接读写。

由于多个进程共享一段内存，依靠某种同步机制（如信号量）实现进程间的同步和互斥。

基于数据结构共享：共享空间内只放一个数组。速度慢、限制多，一种低级通信方式。

基于存储区的共享：OS在内存画出一块共享存储区，数据的形式、存放位置由通信进程控制。速度快，一种高级通信方式。

消息Message队列

在消息传递系统中进行进程间的数据交换，以格式化的消息为单位（计算机网络中把message称为报文）。程序员直接利用系统的提供的一组通信命令（原语）进行通信。OS隐藏通信实现细节，简化通信程序编制的复杂性。
分为直接通信和间接通信。

直接通信方式：发送进程利用OS的发送命令直接把消息发送给目标进程（`Send(Recevier,message); Receive(Sender, message);`）。

间接通信方式：以“信箱”为中间实体进行消传递。

管道pipe

连接一个度进程和一个写进程以实现通信的共享文件（不隶属于文件系统）。

管道只能采取半双工通信（某一时间段内只能实现单向传输），个进程互斥访问管道（由OS实现），任意时刻只有一个进程对管道进行操作。

线程与进程

进程是系统资源分配的单位，线程是处理器调度的单位。

为了让程序能并发执行，系统必须进行：创建进程，撤销进程，进程切换。
进程时进程实体的运行过程，是系统进行资源分配与调度的一个独立单位。将这两个属性由操作系统分开处理，并对拥有资源的基本代为不进行频繁切换，于是形成了线程的概念。

-	**进程**	**线程**
组成	程序段、数据、PCB	共享其隶属的进程，但拥有自己的线程ID、寄存器等
并发性	没有引入线程的系统重，进程是独立运行的基本单位	线程是独立的基本单位，一个进程至少拥有一个线程
资源	进程是资源分配和拥有的基本单位	共享其隶属于进程的资源，但也拥有自己的线程资源
调度	没有引入线程的系统中，进程是独立调度的基本单位	在引入线程的系统重，线程是独立调度的基本单位
通信	信号量，共享存储，消息传递系统，管道通信	同一个进程内的线程直接共享该进程的数据段，不同进程属于进程通信
地址空间	相互独立	同一个进程的各个线程共享该进程地址空间

线程的属性

1. 轻型实体：线程基本不拥有系统资源（只保留必不可少能保证独立运行的资源
2. 独立调度和分派的基本单位（由于是轻型实体，所以调度速度快）
3. 可并发执行
4. 共享进程资源

线程的状态参数（线程独有的且无法共享的）：寄存器状态，堆栈，线程运行状态，优先级，现成转悠存储器，信号屏蔽。

多线程OS中的进程

进程作为拥有系统资源的基本单位，包含多个线程并提供资源（但此时进程不再作为一个执行实体）

多线程OS中的进程包含的属性：作为系统资源分配的单位，可包含多个线程，进程不是一个可执行实体。

线程的实现

用户级线程、内核级线程

用户级线程

进存在于用户控件，创建、撤销、线程间同步与通信等，都无需OS调用。

可以在不支持线程操作系统中实现，创建、销毁、切换等线程管理代价比内核线程少，允许每个进程定制调度算法，管理灵活，能够利用的表空间和堆栈空间比内核级线程多。
但同一进程只能有一个线程同时运行，且如果线程使用系统调用使其被阻塞，整个进程都会挂起。页面失效也会产生类似问题。

内核级线程

在内核支持下运行，创建、撤销、切换都依靠内核实现。为每一个内核支持线程设置了一个线程控制块，内核根据该控制块而感知线程存在，并加以控制。

某一线程被阻塞，其它线程依旧可以并发运行，不同的线程可以进入不同的处理机运行。
但进程切换慢，线程管理开销大。

多模型线程

1. 多对一模型：不需要切换到核心态，系统开销小，效率高；但当一个线程被阻塞后，整个进程被阻塞。
2. 一对一模型：一个线程被阻塞后其它线程依旧继续运行，并发能力强；但一个用户进程会占用多个内核线程，切换到内核态开销大，线程管理成本高。
3. 多对多模型：多对一和一对一的折中。

补充内容

- 设备分配依据设备分配表，不需要创建新进程
- 进程的阻塞或唤醒不是撤销与建立的过程
- 进程自身决定从运行状态到阻塞状态
- P操作(wait(S))可能会导致进程阻塞
- P、V操作实现进程同步，信号量初值由用户决定
- 同一进程或不同进程内的线程都可以并发执行

处理机管理

调度的层次

作业调度（高级调度）、内存调度（中级调度）、进程调度（低级调度）

进程状态切换依赖于调度。调度是为了合理的处理计算机软/硬件资源。

-	高级调度	中级调度	低级调度
功能	按照某种机制，将外存上后备队列中的作业调入内存，为其创建进程、分配资源、插入就绪队列。	内存空间紧张时，将暂时不具备运行条件的进程挂起（调至外存），释放内存资源。内存满足挂起状态的某进程需求时，由中级调度算法决定把外存上的具备运行条件的挂起进程重新调入内存。	将处理机分配给就绪队列中的某进程。是最基本的调度、操作系统核新部分，批处理OS、分时OS、实时OS等的必备功能。
作用	外存→内存（面向作业）	外存→内存（面相进程）	内存→CPU
频率	最低	中等	最高
影响	无→创建态→就绪态	挂起态→就绪态 （阻塞挂起→阻塞态）	就绪态→运行态

调度方式

非抢占式优先调度算法:
系统一旦把处理机分配给就绪队列中优先权最高的进程后，该进程一直执行到完成。

抢占式优先调度算法:
执行期间只要出现另一优先权更高的进程，进程调度程序就立即停止当前进程（原有限权最高的进程），重新分配处理机给新的优先权更高的进程。

调度时机

- 不能进行进程调度、切换的情况
  - 在处理终端过程中
  - 进程在操作系统内核程序临界区中
  - 其它需要完全屏蔽中断的原子操作的过程中
- 应该调度
  - 发生引起调度条件且当前进程无法继续运行
  - 中断处理、自陷处理结束

调度的实现方式

1. 具有高、低两级调度的调度队列模型：按照等待事件的不同，可以放入不同的队列中（不同队列可能有着不同的调度算法）。
2. 具有三级调度时的队列模型：

调度的准则

调度的目的是为了资源利用率高，尽可能保证所有进程的权利是一致的。要求CPU利用率高。

周转时间：作业从提交到系统至完成的时间间隔，周转时间T = 完成时间TF - 提交时间TB

带权周转时间T：Tweight，作业的周转时间T与系统为它提供服务时间Ts之比，W = T/Ts

平均周转时间：所有作业周转时间的平均值，T=(1/n)[iΣ(i=1)Ti]

平均带全周转时间：所有作业周转时间的平均值，W=(1/n)[iΣ(i=1)Ti/Tsi]

调度的算法

调度算法	算法思想	调度类型	优先级
先来先服务	从后背作业队列汇总萱蓦若干最先进入	作业调度&进程调度	等待时间
短作业优先	以运行/计算时间为优先级	作业调度&进程调度	运行时间
优先级调度	以优先数为优先级	作业调度&进程调度	优先数
高响应比优先	响应比=等待事件+运行时间/运行时间	作业调度	响应比
时间片轮转	所有进程按照轮询模式运行	进程调度	轮询模式
多级反馈队列	设置优先级不同的队列里，队列内部按照FCFS规则	进程调度	队列位置

先来先服务FCFS

以等待时间为优先级的调度算法。从后备作业队列中选择若干年最先进入该队列的作业调入内存，为其分配资源、创建进程、放入就绪队列。
利于长作业，不利于短作业；利于CPU繁忙的作业，不利于IO繁忙的作业；可以用于进度、作业调整。

短作业优先SJF

以运行/计算时间为优先级，优先将运行计算时间短的若干作业调入内存。可用于进程、作业调度

利于短作业；比FCFS改善平均周转时间和平均带全周转时间，缩短作业的等待事件，提高系统吞吐量。
但必须预知作业运行时间，对长作业非常不利（长作业的周转时间会明显增长）；人机无法实现交互；完全未考虑作业紧迫度，不能保证紧迫性作业能得到及时处理。

优先级调度算法PSA

依赖外部赋予的优先级（一般假设优先数越大，优先级越小）。可由于进程、作业调度。
优先级在进程创建时被生成，按创建后优先级能否改变分为静态优先级、动态优先级；按更高优先级能否抢占正在执行的进程分为菲薄多事、剥夺式。
优先级设置：系统进程>用户进程，交互型进程>非交互型进程，IO型进程>计算型进程。

高响应比优先调度算法

每次选择作业投入运行时，先计算此时后备作业队列中每个作业的响应比RP然后选择其数值最大的作业投入运行。只用于作业调度。

优先权Rp = 1 + 等待时间/服务时间 = 响应(周转)时间/服务时间。

若作业等待时间相同，则要求服务的时间越短，其优先权越高，此时类似SJF算法利于短作业；
若要求服务时间相同，作业优先权取决于等待时间，此时类似FCFS算法利于长作业；
对于长时间的优先级，可以为随等待事件的增加而提高，当等待时间足够长，也可以获得处理机。

时间片轮转

按照各进程到达就绪队列的顺序，轮流让各个进程执行一个时间片，若进程未在该时间片内执行完则会剥夺处理机，将进程放到就绪队列队尾。分时操作系统常用，可以及时响应用户请求，保证所有进程在面对CPU时的优先级一样。只用于进程调度。

多级反馈队列

动态调整进程优先级和时间片大小。只适用于进程调度。

设置多级就绪队列，各级队列优先级从高到低，时间片从小到大。新锦成到达时先进入第1级队列，按FCFS原则排队等待被分配时间片，若用完时间片后进程还未结束则进入下一级队列队尾，若此时已经在最下级队列对位，则重新放回最下级队列队尾。只有第k级队列为空，才会为k+1级对头的进程分配时间片，被抢占的处理机进程重新放回原队列队尾。

进程的上下文切换

CPU寄存器和程序计数器就是CPU的上下文，都是CPU在运行任何任务前必须的依赖环境。
进程具有共享性、并发性。进程的状态切换依赖于调度。调度需要进程控制块PCB。

进程的上下文切换就是把前一个任务的CPU上下文保存，然后加载新任务的上下文到这些寄存器和程序计数器，最后挑战到程序计数器所指的新位置，运行新任务。

调度实现方式：具有高、低两级调度的调度队列模型；具有三级调度的调度队列模型。

系统调用过程：传递系统调用参数 -> 执行陷入指令（用户态） -> 执行相应的内请求核程序处理系统调用（核心态） -> 返回应用程序

进程由内核管理和调度，进程的切换只发生在内核态。进程的上下文包括虚拟内存、栈、全局变量（用户空间资源），内核对战、寄存器状态（内核空间状态）。
进程的上下文切换比系统调用多了：保存内核态资源前，先把进程的用户态资源保存；加载下一进程内核态后，还需刷新进程的虚拟内存和用户栈。

进程上下文切换的场景：

- 进程时间片耗尽，就会被挂起并切换到其它等待CPU的进程运行
- 系统资源不足，进程要等待资源满足后才能运行
- 进程通过睡眠函数sleep主动挂起
- 有优先级更高的进程运行
- 硬件中断

死锁的概念和性质

计算机有可重用性资源（计算机外设），消耗性资源（数据、消息），可抢占性资源（不引起死锁，CPU、内存），不可抢占行资源（光驱、打印机）。多道程序系统可借助多个进程的并发执行改善系统资源利用率，提

高系统吞吐量，但可能导致死锁。

死锁是当多个进程在程序运行该过程中因争夺资源而造成一种僵局。此状态OS若无外力作用酱无法向前推进。

死锁的原因有：系统资源的竞争（不可剥夺资源），程序推进顺序非法（请求和释放资源的顺序不当，信号量的使用不当），死锁产生的必要条件（互斥条件、不剥夺条件、请求并保持条件、循环等待条件）

死锁产生的必要条件

- 互斥使用：在一段时间内某资源只能由一个进程占用。其它进程请求只能等待直到该资源进程使用完毕将其释放
- 请求保持：进程已占用至少一个资源，有申请被其他进程占用不释放的资源，则会请求进程阻塞，但不释放资源
- 不可抢占：进程已获资源在使用完之前，不能被其他进程抢占，只能在使用完后自己释放
- 循环等待：发生死锁时，必须存在一个进程-资源循环链

死锁的处理策略

静态策略：预防死锁；
动态策略：避免死锁，检测死锁，解除死锁。

安全状态：系统能按照某种进程顺序（安全序列）为每个进程分配其所需资源，直到满足每个进程对资源的最大需求，使每个进程都可以顺利完成（如果系统无法找到这样的安全序列，则称系统处于不安全状态）。

死锁预防

通过实现采取某种限制措施，破坏死锁产生的必要条件，达到预防死锁的目的。

1. 条件一：互斥使用：由设备的固有条件决定，不仅不能改变，还能加以改变。
2. 条件二：请求保持条件：可破坏
    1. 协议1：所有进程开始运行前，必须一次性申请其在整个运行过程中所需的全部资源。运行期间不再申请新资源；等待期间不占有任何资源，破坏“保持”条件；协议简单安全，但会造成资源浪费，经常发生金城街现象
    2. 协议2：进程获得运行初期所需资源后便开始运行。资源利用率高，能有效减少饥饿现象的概率
3. 条件三：不可抢占：可破坏
    - 已经保持了某些不可被强占资源的进程，提出新的资源请求而不能得到满足时，其申请到的资源可以被剥夺，以后需要重新申请。实现复杂，代价大
4. 条件四：循环等待：可破坏
    - 对系统所有资源类型进行现行排序，并赋予不同的序号；进程必须按照序号递增顺序申请资源。对资源安排很重要，系统资源利用率和吞吐率比其它策略有明显改善；资源序号相对稳定性导致新设备的增加受限，由于作业使用资源的顺序和系统资源序号顺序未必相同，可能导致资源浪费。

死锁避免

资源动态分配过程中用某种方式防止系统进入*不安全状态*，避免发生死锁。
避免死锁比预防死锁是假的限制条件弱，较完善的系统常用死锁避免处理死锁，允许进程动态地申请资源，但OS分配资源前要计算资源分配的安全性。

银行家算法

引入：
可利用资源向量Available数组，每一个元素代表一类可以用的资源数目；
最大需求矩阵Max（如果Max[i,j]=K，表示进程i需要Rj类资源的最大数目为K）；
分配矩阵Allocation（如果Allocation[i,j]=K，表示进程i当前已分到的Rj类资源数目为K）；
需求矩阵Need（如果Need[i,j]=K，表示进程i还需Rj类资源K个）。

设Requesti是进程Pi的请求向量，如果Requesti[j]=K，表示进程需要K个Rj类型资源，发出资源请求后将按如下步骤检查：

1. if Requesti[j] ≤ Need[i,j]，转到步骤2；否则出错
2. if Requesti[j] ≤ Available[j]，跳转3；否则Pi需要等待
3. OS试探着把资源分配给进程Pi，并修改下列数值：
    - `Available[j] := Available[j] - Requesti[j];`
    - `Allocation[i,j] := Allocation[i,j] + Requesti[j];`
    - `Need[i,j] := Need[i,j] - Requesti[j]`

随后系统执行安全性算法检查此次资源分配后系统是否出于安全状态，以判断是否需要正式分配给Pi。安全性检查算法步骤如下：

1. 设置两个向量：
    1. 工作向量Work：系统可供给进程继续运行所需的各类资源数目，在执行安全算法开始时，Work := Available
    2. Finish：表示系统是否有足够资源分配给进程，使之运行完成。开始时Finish[i]:=false；有足够资源时Finish[i]:=true
2. 找到一个能满足下述条件的进程（若找到则执行步骤3；否则执行4）
    1. Finish[i]=false;
    2. Need[i,j]≤Work[j]
3. 进程Pi获取资源并顺利执行完成后，释放资源，执行：
    - `Work[j] := Work[i] + Allocation[i,j];`
    - `Finish[i] := true;`
    - *`go to step 2;`*
4. 如果所有进程Finish[i]=true都满足，表示系统处于安全状态；否则处于不安全状态

死锁检测

通过系统设置的检测机构来及时检测死锁的发生，精确确定与死锁有关的进程和资源，清除已发生死锁的进程。

引入资源分配图描述系统死锁情况，G=(N,E)，G为图，N表示G的顶点集合，E表示G的边集合。

- 检测定理：利用简化资源分配图检测当前系统是否处于死锁状态
    - 在资源分配图中找出一个既不阻塞也不独立的进程节点Pi
    - 顺利时，Pi可获得所需资源继续运行，直到完成时释放其占有的全部资源，相当于消去Pi请求边和分配边，使之成为孤立节点

死锁解除

检测到系统已发生死锁时，需将进程从死锁状态中解脱。
它和死锁检测相配套。通过回收资源并再分配给处于阻塞状态的进程，是指转为就绪状态继续运行。

解除方法有：

- 抢占资源：强行剥夺部分死锁状态进程所占资源，并分配给其它死锁状态的进程使之能够运行，从而解除死锁
- 撤销/挂起/终止进程：
  - 终止所有死锁进程（简单但代价大）
  - 逐个终止进程：一次选择付出代价最小的进程并终止，直到有足够资源打破循环等待，并将系统从死锁状态解脱
- 付出代价最小的死锁接触算法：经解除进程的代价按升序排列，每次终止队首进程，直到死锁解除

补充知识点

- 进程调度算法采用固定时间片轮转调度算法时，时间片过大会使得时间片轮转发算法转化为FCFS算法
- FCFS算法有利于CPU繁忙型作业，不利于IO繁忙型作业
- 所有进程同时到达时，进程平均周转时间最短的事短进程优先算法。短进程算法是性能最好的算法
- 综合考虑进程等待时间和执行时间的是高响应比算法。响应比 = （等待时间+服务时间)/服务时间
- 满足段任务有限且不会发生饥饿现象的调度算法是高响应比优先算法

-	**先来先服务**	**高响应比优先**	**时间片轮转**	**非抢占式短任务优先**
优先短任务?	×	√	×	√
会发生饥饿?	×	×	×	√

- 计算算法性能的表格：

任务	**提交时间**	**运行时间**	**开始运行**	**结束运行**	**等待时间+服务时间**	**响应比**

- 有n太互斥使用的同类设备，三个并发进程需要3、4、5太设备，可确保不发生死锁的设备数为2+3+4+1 = 10
- 死锁预防会限制用户申请资源的顺序，死锁避免不会

内存管理

存储器程序思想：指令和数据以同等地位存于存储器；指令和数据用二进制表示。存储器用来存放数据和程序。
主存储器的编制单元是字节，内存地址也叫物理地址。

内存性能指标：存储容量、单位成本，存储速度。
存储容量 = 存储字长*存储字数
每位价格 = 总成本/总容量
传输速率 = 数据带宽/存储周期

内存层次结构（由快到慢）：寄存储器，缓存，主存，磁盘，光盘，磁带。
其中缓存-主存解决速度问题，主存-辅存解决容量问题。

基本概念

实现内存的分配、回收、保护、扩充。

- **逻辑地址LA**：程序经过汇编或编译后，形成目标代码，每个目标代码都以0为机制顺序进行编址，原来用富豪名访问的单元用具体的数据*单元号*取代。这样生成的目标程序占据一定地址空间，称为作业的逻辑地址空间（逻辑地址）
- **物理地址PA**：内存由若干个存储单元组成，每个存储单元有一个唯一标识一个存储单元的编号，称为内存地址（物理地址） 把内存看成一个从0字节一直到内存最大容量诸子阶编号的存储单元数组（每个存储单元与内存地址的编号相对应）
- **内存映射**：逻辑地址 -> 物理地址
- **内存保护**：内存分配前，保护操作系统不受用户进程影响，同时保护用户进程不受其他用户进程的影响
    - *硬件实现*：在CPU内设置一对下限寄存器和上限寄存器，存放正在执行的程序在主存中的上限和下限地址。每当CPU访存，硬件自动将被访问的主存地址与节点寄存器进行比较，判断是否越界。若无越界，则按此地址访问主存，否则产生程序中断（存储保护中断
- **内存分配和回收**：程序执行时分配空间，不执行时则回收
- **内存扩充**：物理上对内存扩充（插内存条），软件上使用虚拟内存

程序的运行过程

编译->链接->装入->执行
编写程序代码，*编译*代码并产生目标模块，通过*链接*程序到*装入*模块，放入程序，最后到内存中*执行*
过程可分为四个部分：预处理器->编译器->汇编器->连接器

运行过程

1. 预处理：
    1. 处理所有注释以空格代替
    2. 将所有#define删除，并展开所有宏定义
    3. 处理条件编译指令#if, #ifdef, #dlif, #sles, #endif
    4. 处理#include，展开文件包含
    5. 保留编译器需要用#program指令
2. 编译：将经过预处理后的程序转换成特定汇编代码

3. 汇编：将汇编代码转为机器码，产生二进制的目标文件
4. 链接：将有关目标文件相连接，即将在一个文件中引用符号同该符号在另一个文件中定义连接起来，使所有这些目标文件能成为一个能让操作系统装入执行的统一整体

链接方式

静态链接：程序执行前，将若干0地址开始的目标模块及所需库函数链接为完整、从唯一“0”地址开始的装配模块。

装入时动态链接：边装入内存边链接个目标模块。

运行时动态装入：程序执行过程中需要目标模块时，才对其进行链接。

装入方式

绝对装入方式：编译程序产生的绝对地址的目标代码，程序中的逻辑地址与实际内存地址完全相同；在编译时，如果知道程序将放到内存中的位置，编译程序将产生绝对目标地址的代码。
装入程序按照装入模块中的地址，将程序和数据装入内存。只适用于单道程序环境。程序中使用的绝对地址，可在编译或汇编时给出，也可以由程序员直接赋予。通常情况下都是编译或汇编时再转换为绝对地址。

可重定位装入方式：多道程序环境下，装入一次性完成逻辑地址到物理地址的变换，并根据内存的当前情况将装入模块装入到内存的适当位置（又称静态重定位）。
静态重定位必须分配其要求的全部内存空间，如果没有足够内存则不能装入改作业。作业一旦进入内存后，运行期间不能再移动、也不能申请内存空间。

运行时动态装入：装入内存后所有地址仍为相对地址，地址变换在程序执行期间、随着对每条指令的访问自动进行（又称动态重定位）。
编译、链接后的装入模块地址都从0开始。装入程序把装入模块装入内存后并不立即把逻辑地址转为物理地址，装入内存后所有地址依然是逻辑地址（地址转换等到程序真正要执行时才进行）。

连续内存的管理方式

连续分配：单一连续分配，固定分区分配（内部碎片），动态分区分配。

- 动态分区分配
  - 外部碎片、紧凑
  - 分配策略：
    - 首次首映算法：地址递增
    - 最佳适应算法：容量递增
    - 最坏适应算法：容量递减
    - 邻近适应算法
  - 回收：回收块与相邻空闲块合并；空闲块再排序

内碎片：占用分区之内未被利用的空间；
外碎片：占用分区之间难以利用的空闲分区（通常是小空闲分区）

单一分配方式

把内存分为系统区（仅供OS使用，内存的低址部分）、用户区（系统去外全部内存空间）。最简单的存储管理。

实现简单；无外部碎片；可采用覆盖技术扩充内存；不一定需要采取内存保护。
但只用于单用户、单任务操作四天；有内部碎片；存储器利用率低；用户区在小作业时浪费。

固定分配方式

将用户控件华为若干个固定大小的分区，每个分区中只装入一道作业，形成一种最简单的可运行多道程序的内存管理方式。（有分区大小相等和不等两种形式）。

分配流程：检索分区使用表：

- 存在满足用户程序大小且未被分配的分区：
  - 用户程序装入满足条件的分区
  - 分区使用表：占用一个空白表项（用户程序大小，起始地址，分区状态：为已分配）
- 不存在满足条件的分区：拒绝分配内存

操作系统需要建立一个数据结构：分区说明表，来实现每个分区的分配与回收。每个表项对应一个分区，通常按分区大小排列。每个表项包括对应分区大小、起始地址、状态（是否已分配）。由内核态管理。

实现简单，无外部碎片。
但当用户程序太大时，可能所有分区都不能满足需求，此时不得不采用覆盖技术解决，又会降低性能；会产生内部碎片，内存利用率低。

动态区分分配方式

在进程装入内存时根据进程大小动态地创建分区，并使分区大小正好适合进程的需要。因此系统分区大小和数目是可变的，又称可变分区分配。

动态分配算法有：

- 首次适应法FF：顺序找到一满足的就分配，但可能存在浪费
  - 空闲分区一地址递增的次序链接，分配内存是顺序查找，找到大小能满足要求的第一个空闲分区。
  - 目的在于减少时间。
  - 空闲分区表（空闲区链）中的空闲分区要按地址由高到低排序。
- 循环首次适应发NF：类似哈希算法中左右交叉排序，满足就分配
  - 有首次适应算法演变。分配内存从时从上次查找结束的位置开始继续查找。
  - 空闲分区分布更均匀，查找开销小。
  - 从上次找到的空闲区的下一个空闲区开始查找，直到找到第一个能满足要求的空闲区位置，并从中划出一块与请求大小相等的内存空间分配给作业。
- 最佳适应算法BF：找到最合适的，但大区域访问次数少
  - 空闲分区按容量递增形成分区链，找到第一个能满足要求的空闲分区。
  - 能使外碎片尽量小。外部碎片较多。
  - 空闲分区表（空闲区链）中的空闲分区要按大小从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。
- 最坏适应算法WF：相对于最好而言，找到最大的区域下手，导致最大区域可能很少，也造成许多碎片
  - 空闲分区以容量递减的次序链接，找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。
  - 外部碎片较少，但可能没有足够多的空间留给未来的较大程序。
  - 空闲分区按由大到小排序。

连续分配算法总结：

动态分区	算法思想	性能分析	内部碎片	外部碎片
首次适应法	地址递增，满足大小的第一个分区	效果最好	无	有
最佳适应算法	空闲分区按容量递增，满足大小的第一个分区	效果最差、外部碎片最多	无	有
最坏适应算法	空闲分区一容量递减的次序链接，满足大小的第一个分区	缺少大分区	无	有
临近适应算法	地址递增，分配内存时从上次查找结束的位置开始继续查找	碎片分布整个区间	无	有

静态分区	算法思想	性能分析	内部碎片	外部碎片
单一连续分配	内存分为系统区和用户区	只能单道运行	有	无
固定分区分配	内存事先分为不同的区域	可能无法存放较大程序	有	无

基本分页内存管理方式

在连续存储管理方式中，固定分区会产生内部碎片，动态分区会产生外部碎片——对内存利用率都较低。内存中可能缺少大块连续空间。
分页存储管理将一个进程的逻辑地址空间分成若干个大小相等的片，称为页面或页，并为各页编号，从0开始。
把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框frame，甲乙编号，从0#块、1#块等开始。

基本概念：

- 页面：将一个进程的逻辑地址空间分成若干大小相等的片
- 页框frame：内存空间分成与页面相同大小的存储块
- 页内碎片：进程最后一项经常装不满一块而形成的不可利用碎片
- 地址结构：页号P+位移量W(0-31)
- 页表：让进程能够在内存中找到一个页面所对应的物理块，系统为每个进程建立的页面映像表。实现从页面号到物理块号的地址映射

*页框、块、页块、帧、页帧、根、叶根*是一个东西

逻辑地址结构

在为进程分配内存时，一块为单位将进程中的若干个页分别汉族昂入到多个可以不相邻接的物理块中。
分页地址中的地址结构：`|页号P|位移量W|`，
eg.`|31--12|11--0|`：页面大小：`2^W B`，页面大小：`2^12 B`，页数：`2^P 个`，页面个数：`2^20 个页面`。

对于特定机器，其地址结构是一定的。若给定地址为A，页面大小L，页号P和页内地址d公式为：
`P = int(A/L)`，`d = A%L`。

页表

一个进程对应一张页表。进程的每个页面对应一个页表项；每个页表项“块号”和其他信息组成。每个页表项的长度相同。页表项大笑等于也狂所占的比特数。页表纪录进程页面和实际存放内存块之间的映射关系。页表本质上是一个大数组，页号是数组小标，页表项是数组元素，其大小是块号。

地址转化过程

基本的地址变换机构：要访问两次内存；页表大都驻留在内存中；为了实现地址变换功能，在系统中的设置页表寄存器PTR，用来存放页表的始址和页表长度。在进程未执行时，每个进程对应页表的始址和长度都存放在进程的PCB中，当该进程被调度时，就将他们装入页表寄存器。

地址转换的过程如下：

1. 程序执行时，从PCB去除页表的始址和页表常茹，装入页表寄存器PTR
2. 由分页地址变换机构将逻辑地址自动分成页号和页内地址
3. 将页号与页表长度进行比较，若页号≥页表长度，表示本地访问的地址已经超过进程的地址空间，产生越界中断
4. 将页表始址与页号和页表项长度的乘积相加，便得到该页表项中的位置
5. 取出页描述子得到该页物理块号
6. 对该页的存取控制进行检查
7. 将物理块号送入物理地址寄存器中，再将有效地址寄存器中的业内地址直接送入物理地址寄存器的快内地址字段中，拼接得到实际的物理地址

处理器每访问一个内存中的操作数，就要访问两次内存：第一次查找页表将操作数的逻辑地址变换为物理地址；第二次完成真正的读写操作。

具有快表的地址转化过程

为了缩短查找时间，将页表从内存装入CPU内部的关联存储器（例如快表），实现按内容查找。
此时的转化过程：CPU给出有效地址后，由地址变换机构自动将页号送入快表，并将此页号与快表中的所有页号进行比较。若有相匹配的页号，表示要访问的页的页表项在快表中。于是可以直接读出该页对应的物理页号，无需访问内存中的页表。（访存次数降低为一次）

二级页表

现代大多数计算机系统都支持非常大的逻辑地址空间（2^32~2^64），页表就变得非常大，并且要求是连续的。于是有两种解决方式：
1，离散分配方式；
2，只将当前需要的部分页表项调入内存，其余页表仍驻留在磁盘上，需要时再调入。

二级页表的逻辑地址结构可描述为：
|外层页号|内层页号|页内地址|
由外部页表去查找内部页表，内部页表的每一项都会对应内存空间中的一块。
（内存的管理单位为块，外部页表与内部页表存储的是块号，任意一级页表大小不得超过一页(块)大小）。
访存三次：页目录表、二级页表、存储单元。

多级页表

对于64位机器，采用二级页表已经不够用了。将外层页再进行分页，将各分页离散地装入不相邻接的物理块，再利用第二级的外层页表来映射它们之间的关系。逻辑地址结构可表述为：

|1级页表|2级页表|3级页表|4级页表|5级页表|页内地址|
访问内存次数=N+1次。

基本分段内存管理方式

分段存储管理方式是为了满足用户和程序员*方便编程*、*信息共享*、*信息保护*、*动态增长*、*动态链接*的需求而引入。

分段的地质结构

|段号|段内地址|
作业的地址空间悲哈氛围若干段，每个段都是一组完整的逻辑信息，每个段都有自己的名字，都是从零开始编址的一段连续地址空间，各段长度不等。
内存悲动态地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。

段表

作业空间指向段表对应内容，段表的每一项都指向一个内存空间对应段。

地址变化过程

1. 程序执行时从PCB取出段表始址和长度并装入段表寄存器
2. 地址变换机构将逻辑地址自动分成段号和段内地址
3. 将段号和段表长度比较，若段号≥段表长度，表示本次访问的地址超越进程的地址空间，产生越界中断
4. 取出段描述子得到该段的段表始址和段长
5. 将段内地址和段长比较
    - 若段内地址≥段长，表示本次访问的地址超越进程的地址空间，产生越界中断
6. 将段表始址和段内地址相加得到物理地址

分段和分页的区别：
页是信息的物理单位。页的大小固定且由系统固定。分页和用户程序地址空间是一维的。段通常比页大，段表比页表短，可以缩短查找时间，提高访问速度；分页是系统管理的需要，分段是用户应用的需要。一条指令或一个操作数可能会跨越两个页的分解诶很粗，而不会跨越两个段的分界处。

虚拟存储器管理方式

具有请求调入功能和置换功能，从逻辑上对内存容量扩充的一种存储器系统。逻辑容量由内存和外存容量之和决定。

- 特点：
    - 离散性，多次性，对换性，虚拟性
- 优点：
    - 大程序：可在较小的可用内存中执行较大程序
    - 大用户空间：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
    - 以CPU时间和外存空间换区昂贵的内存空间
- 实现方式：
    - 硬件，请求分页中的内存分配，物理块分配算法，页面调入策略

常规存储器的管理方式特征：
驻留性：作业被装入内存就会一直驻留在内存中知道运行结束。但实际上只要访问作业的一小部分数据就能正常运行。
时间局部性：程序中存在大量循环操作，使得指令和数据被多次访问。
空间局部性：程序在一段时间内访问的地址可能集中在一定范围内（多见于程序顺序执行和数组）

请求分页管理方式

请求分页的代表机制，重要机构，地址变换

1. 硬件支持：
    1. 页表机制：纯分页的页表机制上增加若干项而形成作为请求分页的数据结构
    2. 缺页中断机构：产生缺页中断时请求OS将所缺页调入内存
    3. 地址变换机构：纯分页地址变换机构的基础上发展形成
2. 请求分页的软件

页表机制

结构如下：
|页号|物理块号|状态位P|访问字段A|修改为M|外存地址|

页号：虚拟地址空间中的页号；
物理块号：该页所占内存的块号；
P：1表示在内存中，0反之且会发生缺页中断信号；
A：系统规定时间内该页是否被引用过；
M：1表示被修改过，0表示未被修改过；
外存地址：该页内容在辅存中的地址，缺页时将所缺页调入内存。

缺页中断机构

1. 处理过程：
    1. 根据当前执行指令中的虚拟地址，形成(页号,页内偏移)，用页号查页表，判断该页是否在内存中
    2. 若缺页中断位为0（页面不在内存），产生缺页中断，让OS中断处理程序进行中断处理
    3. 查询存储分块表，寻找空闲内存块；查询页表得到该页在辅存中的地址，启动磁盘读信息
    4. 把磁盘上读出的信息装入分配的内存块
    5. 根据分配存储块的信息，修改页表、存储分块表汇总相应表目的信息
    6. 完成所需页装入工作后，*返回原指令*重新执行
2. **缺页中断和一般中断的区别【重点】**：
    1. 缺页中断在执行一条指令中产生，并且立即处理；一般中断是在指令执行完毕后响应和处理
    2. 缺页中断处理执行完后，仍返回到原指令处重新执行；一般中断则返回下一条指令执行
    3. 一条指令执行过程中可能会发生多次中断

地址变换机构

将虚拟地址变换为物理地址。流程如下：

1. 请求调页：查到页表项时进行判断
2. 页面置换：需要调入页面，但没有空闲内存块时进行
3. 需要修改请求页表中新增的表项

逻辑地址LA -> PA 映射情况的可能形式：

1. TLB+内存
2. TLB+页表+内存
3. TLB+页表+中断+TLB+内存

页面置换算法

按照某种机制把一部分页面淘汰出去，把虚拟内存中的页面加载出来。

-	**算法规则**	**优缺点**
OPT	有限淘汰最长时间内不会被访问的页	缺页率最小，性能最好； 无法实现
FIFO	优先淘汰最先进入内存的页	实现简单；性能差，可能 出现Belady异常
LRU	有限淘汰最近最久未访问的页	性能好；但要硬件支持， 算法开销大
简单 CLOCK	循环扫描，淘汰访问位为0的，并将扫描过的页面置为1.第一轮未选 中则进入第二轮	实现简单，算法开销小； 未考虑页面是否被修改过
改进 CLOCK	引入(访问位,修改位)。第一轮淘汰(0,0)；第二轮淘汰(0,1)，置扫描 过页面的访问位为0；第三轮淘汰(0,0)；第四轮淘汰(0,1)	算法开销小，性能较好

最佳置换算法OPT

其所淘汰的页面僵尸以后永不使用、或许是在最长未来时间内不再被访问的页面。理论上可保证获得最低的缺页率，但实际上是无法实现的。
只有在进程执行过程中才能知道未来会访问到那个页面，OS无法提前预判页面访问序列。
不过可以以此为标准评估其它算法。

先进先出算法FIFO

其所淘汰页面是最早进入内存的页面。
把内存页面根据调入先后顺序排成一个队列，选择队头页面换出。队列的最大长度取决于OS为进程分配的内存块数。

Belady奇异现象：采用FIFO时，若对一个进程未分配它所需的全部页面，有时会出现分配页面数增多，缺页率反而提高的异常现象。
因为FIFO未遵循程序局部性原理，完全以进入时间为基准来判断。

最近最久未使用置换算法LRU

其所淘汰页面是最近最久未使用的页面。
赋予每个页面的对应页表项中，用访问字段纪录该页面自上次被访问以来所经历时间t。选择现有页面中t最大的淘汰（此页面即为最近最久未使用的页面）。

为了记录进程在内存中各页的使用情况，需要为每个在内存中的页面配置一个位移寄存器。
效率高，开销大。

简单CLOCK算法

为每个页面设置一个访问位A，再将内存中的页面通过链接指针结成一个循环队列。被访问页的访问位A置为1。
淘汰位时检查访问位A：如果为0，该页换出；如果为1，置为0，暂不换出，继续检查下一页面。第一轮扫描中若都为1，则进行第二轮扫描。淘汰页面时最多经过两轮扫描。

但当缓冲池被填满且访问1时就可以直接命中缓存返回。简单CLOCK无法展现这一点。

改进型CLOCK算法

在简单CLOCK基础上引入修改位M。于是可组成一下四类页面：

1. `A=0, M=0`：未被访问、未被修改，最佳淘汰页
2. `A=0, M=1`：未被访问、已被修改，次佳淘汰页
3. `A=1, M=0`：已被访问、未被修改，可能再被访问
4. `A=1, M=1`：已被访问、已被修改，可能在被访问

算法规则：将所有可能被置换的页面排成循环队列。
第一轮从当前位置扫描到第一个(0,0)帧用于替换，不修改任何标志位；
若第一轮失败则进入第二轮，扫描到第一个(0,1)帧用于替换，将所有扫描过的访问位A置为0；
若第二轮扫描失败则进入第三轮扫描到第一个(0,0)用于替换，不修改任何标志位；
若第三轮失败则进入第四轮扫描到第一个(0,1)帧用于替换。
由于第二轮将所有访问位设为0，则在第三、第四轮必定会有一个帧被选中，因此淘汰页时最多会经过四轮扫描。

两个重要概念

1. **系统抖动/颠簸**：请求分页存储管理中，从驻村刚一走一个页面，根据请求又立刻又调入该页，出现反复调入调出的颠簸现象。
    - 若一个进程在换页上用的时间多余执行时间，则发生颠簸
    - 解决方案：增加工作集大小；选择不同的淘汰算法尽量使得工作集页面在内存中
2. **工作集/驻留集**：某段时间间隔内，进程要访问的页面集合。经常被使用到的页面要在工作集中，而长期不背使用的页面要从工作集中丢弃。为了防止系统抖动，需要选择合适的工作集大小。
    - 工作集会根据窗口尺寸算出
    - 只要划分合理的工作集，就不会发生抖动

页面的分配和调入

最小物理块数确定

- 能保证进程正常运行所需的最小物理块数
- 当OS为进程分配的物理块数少于此数值，进程无法运行
- 进程应获取的最小物理块数与计算机硬件结构有关，取决于指令格式、功能、寻址方式

物理块的分配策略

- 固定分配：OS为每个进程分配一组固定数目的物理块，在进程运行期间不再改变（驻留集不变）
- 可变分配：先给每个进程分配一定数目物理块，运行期间可视情况增减（驻留集大小可变）
- 局部变换：发生缺页时只选进程自己的物理块进行置换

- 全局置换：可将操作系统保留的空闲物理块分配给缺页进程，也可将别的进程持有的物理块置换到外存并在分配给缺页进程

在请求分页OS中，可采取固定和可变分配策略。在进行置换是，可采取全局置换和局部置换

1. 固定分配局部置换
    - OS为进程分配一定数量物理块，运行期间不变，发生缺页从内存中的页面换出
    - 缺点：很难在刚开始确定分配数量是否合理
2. 可变分配全局置换
    - 动态分配，物理块用完时可置换其它进程页
    - 当系统空间被分配完，可在整个空间中随意置换
3. 可变分配局部变换
    - 置换自身进程页，频繁缺页时动态分配

何时调入页面

1. 预调页策略：进程开始时，根据局部性原理，一次调入若干个相邻页面。常用于首次调入，由程序员指出先调入部分
2. 请求调页策略：进程运行期间发现缺页时将所缺部分调入内存。但IO开销较大

从何处调入页面

请求分页系统中的外存分为*存放文件的**文件区***和*用于存放对换页面的**对换区***。文件区离散分配，对换区连续分配。

1. 当系统中有足够对换区空间：将全部从对换区调入所需页面。进程运行前将与该进程有关的文件从文件区拷贝到对换区
2. 缺少足够对换区空间：不会被修改的文件直接从文件区调入；换出页面时，由于未被修改而不必将它们换出，以后调入扔从文件区调入。可能被修改的部分在将其换出时需调到对换区
3. UNIX方式：访问过的都在对换区，未访问过的都在文件区

页面调入过程

- 当程序所访问页未在内存，向CPU发出缺页中断
- 程序查找页表得到物理块后，若此时内存能融安新野，则启动磁盘IO所缺页调入内存，然后修改页表
- 如果内存已满，按某种置换算法从内存中选出一页换出：
    - IF未被修改，不必协会磁盘
    - IF已被修改，协会磁盘
- 缺页调入内存后，利用修改后的页表形成要访问的数据物理地址，再访问内存

补充

- 将逻辑地址转变为内存的物理地址的过程称作：重定位（静态重定位、动态重定位）/装载/装入
- 虚拟内存管理中，地址变换机构将逻辑地址变换为物理地址，形成该逻辑地址的阶段是：编译
    - 编译 形成逻辑地址
    - 装载 形成物理地址
- 起始地址为0的目标模块A、B、C，长度依次为L、M、N，采用静态链接方式链接在一起后，模块C的起始地址为L+M
- 内存保护由硬件机构完成，保证进程空间不被非法访问

- 页式存储管理系统中，若页大小4KB，页号0对应块号2，地址转换机构将逻辑地址0转换为物理为：页号0，DIW 4KB = 0，页内地址 0%4KB=0，`4KB*2=4096*2=8192`
- 若二级页表的虚拟地址结构为：`|10位|10位|12位|`。页大小=2^12B=4KB，页大小=页框大小，页数=2^20。假设页目录项和页表项占4字节，总空间大小为`1024*4B+(1024*4B0*1024)`，页数=`总空间大小/(1024*4B)=1025页`。
- 若逻辑地址为(0,137)，对应段表内容`|段号0|内存始址50K|段长10KB|`，其物理地址为`50K+137`。若逻辑地址为(3,4000)，对应段表内容`|段号3|内存始址80K|段长1KB|`，其会产生越界
- 每页1KB，表示页内地址10位；若2页对应物理块号位4，对于逻辑地址`0A5C`，将其转为二级制`0000 1010 0101 1100`，保留`10 0101 1100`，`0000 10`为2页，对应物理块号5，所以其物理地址为`100 10 0101 1100`

文件管理

文件概览

文件是存放于外存的、是文件系统的最大单位文件，是只有创建者所定义的、具有文件名的一组相关元素的组合。现代OS通过文件系统来组织管理计算机外存存储的程序和数据。

目录=文件夹，用于表示系统中的文件及其物理地址的数据结构，对文件进行有效管理，供以检索使用。

文件的术语

- 纪录：表述对象某方面的属性
- 关键字：唯一标识记录的数据项
- 数据项：描述一个对象某种属性的字符集。最低级的数据组织形式

文件的分类

- 按文件性质和用途分类
  - 系统文件：OS软件构成的文件
  - 用户文件：用户的源代码、目标代码、可执行文件或数据结构等构成的文件
  - 库文件：由标准子程序以及日常程序等构成的文件
- 按按文件数据形式分类
  - 源文件：用户的源程序和数据等构成的文件
  - 目标文件：用户源程序经汇编或编译后的文件
  - 可执行文件：目标代码经链接后生成的文件
- 按文件存取属性分类
  - 只执行文件：进允许核准用户调用执行的文件
  - 制度文件：进允许文件主以及核准用户读取的我呢间
  - 读写文件：进允许文件主以及核准用户读写的文件
- 按文件组织形式分类
  - 普通文件：ASCII码或二进制的文件
  - 目录文件：管理文件、实现文件系统功能的系统文件
  - 特殊文件：IO设备文件

文件的属性和操作

属性：类型、长度、物理位置、创立时间。

操作：创建、删除、读取、写入、设置文件读写位置、截断文件；
向上提供的最基本功能（调用系统）：创建create、删除delete、读read、写write、打开open、关闭close。

文件系统模型

用户（程序） -->> |文件系统接口|对对象操纵和管理的软件集合|对象及其属性|

1. 文件管理系统的管理对象：文件、目录、磁盘存储空间
2. 对对象操纵和管理软件的集合：对文件存储空间管理、对文件目录的管理、将文件的逻辑地址转为物理地址的基址、对文件的读写管理、对文件的共享与保护

3. 文件系统接口：命令接口、程序接口

对文件的管理

对文件的操作、逻辑结构、共享、保护、分配方式、目录，对磁盘的组织与管理

文件打开与关闭

打开：OS调用open，将指名文件属性（包括文件在外存上的物理位置）从外存靠背到内存打开文件表的一个表目中，并将该表目的编号（索引）返回给用户。

关闭：OS调用close，将指名文件关闭并把该文件从打开文件表中的表目上删除。

文件的逻辑结构

文件是由创建者所定义的、具有文件名的一组相关元素的组合，可分为无结构文件和有结构文件（顺序文件、记录寻址、索引文件、索引顺序文件、直接文件和哈希文件）。

无结构文件被大量的源程序、可执行文件、库函数所构成（流式文件，以字节为单位）。对流式文件的访问，采用读写指针指出下一个要访问的字符。
有结构文件被大量的数据结构和数据库所采用。

- 顺序文件：有一系列定长或不定长的纪录按某种顺序排列成的文件
  - 串结构：记录按照存入时间排列
  - 顺序结构：记录按关键字有序排列
  - 特点：某个记录的逻辑位置可由第1个记录的逻辑位置推算；查找和存取效率高，但增删改查单个记录难度大。
- 索引文件：问文件创建索引，索引表本身是定长记录的顺序文件，可以快速找到第i个记录对应索引项
  - 可用关键字作为索引号内容（此时还支持折半查找）。当要增删一个记录时，需要对索引表修改
- 索引顺序文件：分为若干组的文件记录，每组第一个记录在索引表中占据一个表项，最常见的逻辑文件形式
  - 索引表：一个顺序文件对应一个表
  - 含N个记录的文件，顺序文件平均查找长度为N/2，一级索引顺序文件查找长度√(N)
- 直接文件：根据给定记录键值直接获得指定记录的物理地址
- 哈希文件：根据键值由哈希查找来定位目录表中的位置

目录管理

目录/文件夹 是用于标识OS中的文件以及其物理地址的数据结构，是对文件进行有效管理的方法，供检索使用。目录之下是目录项。

目录管理的要求：实现按名存取，提高对目录的检索速度，文件共享，允许文件重名。

文件控制块

一个文件控制块FCB就是一个文件目录，对应一个文件夹。目录就是FCB的有序集合。FCB包含的信息：

1. 基本信息类：文件名，文件物理位置（包括存放文件的设备名、在外存上的起始盘块号、文件占用的盘块数or字节数），文件逻辑结构，文件物理结构
2. 存取控制信息类：文件主的存取权限、核准用户的存取权限以及一般用户的存取权限

3. 使用信息类：文件建立日期和时间、上一次修改的日期和时间、当前使用信息（包括已打开该文件的进程数、是否被其它进程锁住）

文件目录的组织方式

目录管理包括有：单级文件目录，两级文件目录，树形结构目录

- **单级文件目录结构**：只建立一个目录表，每个文件占一个目录项（相当于只建一个文件夹）
    - 简单但检索慢，文件不许重名，不便于共享，适合单用户环境
- **两级文件目录**：
    - 每个用户对应一个单独用户文件目录UFD，由该用户所有文件的文件控制块组成
    - 系统建立一个主文件目录MFD，每个用户目录文件占有一个目录项，韩用户名和指向用户目录文件的指针
    - 检索速度较快，不同用户可重名、可用不同名访问同一个共享文件，系统开销大。但缺乏灵活性，无法实现文件分类。不同用户可使用不同文件名来访问系统重的同一个共享文件。
- **树形结构目录**：现代操作系统中最通用且实用的文件目录结构
    - 根节点：主目录/根目录。叶子结点：数据文件。非根非终端节点：子目录
    - 从根目录到任何数据文件只有条唯一通路。对于一个文件，从根目录开始，把途径所有目录文件名与数据文件依次用用“/”连接，形成路径名
    - 每访问一个文件都要使用从树根开始直到树叶为止的、包括个中间节点的全部路径名，效率较低
    - 可以把从当前目录开始直到数据文件为止的路径名，引入为一个相对路径名，简化查找层次，提高效率

目录操作

- 创建目录：用户可为自己建立UFD、创建子目录。创建新文件时，需检查所在目录有无重名文件
- 删除目录：
    - 空目录直接删
    - 非空目录：
        - 不删除非空目录，需要先删除目录中的所有文件使其成为空目录，再删除
        - 若目录中还包含子目录，采用递归调用的方式将其删除
        - 可删除非空目录：删除目录时包含目录中的所有文件和子目录也被同时删除
- 目录检索：
    1. 线性检索法：给定一个目录结构，依次逐级查找
    2. Hash检索法：引入一个Hash目录表，查找目录时查找对应的目录项。若在目录表的相应目录项中的文件名与指定文件名不匹配或发生冲突，此时将Hash值再加上一个常数，形成新的索引值，再返回第一步重新查找

寻找文件

给定一个路径：
用户视角是逐级打开目录访问文件；
OS视角是逐级指向目录，并在最后指向FCB（对应文件的文件控制块）

将目录所在盘块调入内存，嗲查找文件名和目录中的文件名逐一比较：成功则将文件所在盘块调入内存/将信息写入文件所在盘块；失败则表示文件不存在

索引节点

FCB包含有：文件名，文件物理位置，文件逻辑结构，文件物理结构。
在检索信息时仅使用到了文件名，直到找到文件后，才从该目录项中读出其物理地址。于是可以把文件名与文件描述信息分开，使文件描述信息单独形成一个称为索引节点的数据结构，简称为i节点。

在文件目录中的每个目录项仅有文件名和指向该文件所对应i节点的指针构成。

磁盘索引节点：文件主标识符，文件类型，文件存取权限，文件物理地址，文件长度，文件连接计数，文件存取时间。
内存索引节点：文件被打开时，磁盘索引节点拷贝到内存中的索引节点，此时增加了：索引节点标号，状态，访问计数，逻辑设备号，链接指针。

文件的物理结构

文件在存储介质上的组织形式

- 磁盘块：外存中为了方便对文件数据管理，我呢间逻辑地址空间被分为一个个文件“块”
- 内存块：内存管理中，页式管理方式将内存分为块大小
- 文件的逻辑地址可表示为(逻辑快好，块内地址)的形式

文件分配方式：连续分配，文件分配表FAT，索引分配

连续组织方式

连续分配方式。每个文件存放在磁盘的一个刺刀或同一柱面的一组相邻盘快。目录项的“文件物理地址”字段记录该文件第一个记录所在的盘块号和文件以盘块为单位的长度信息。
这样保证了逻辑顺序和存储顺序是一致的，记录该文件第一个记录所在的号和文件长度。这种盘块的相邻关系，体现了逻辑上的位置相邻。

顺序访问容易，顺序访问速度快。
但要求有连续的存储空间，必须事先知道文件长度。

链接组织方式

文件逻辑结构通过链接指针提现，逻辑上相邻的位置通过物理上的上下块之间的指针来体现相邻文职关系。

1. 隐式链接：在每个盘块里，存储了数据、指针（用于指向下一块）
    - 只适用于顺序访问，随机访问效率低。如果一个指针出现问题，整个链会断开
    - 可将同一磁道（柱面）上相邻几个盘块组成一个簇cluster，分盘块时以簇为单位
2. 显示链接：有两个存储结构：磁盘、FAT表
    - 每个表项中存储链接指针。FAT表和磁盘项数一样多
    - 凡是属于某一文件的盘块号，每一条链的链首指针对应的盘块号，均作为文件地址被填入相应文件的FCB的“物理地址”字段中
    - 检索速度快，访问磁盘次数小。但不支持高效的直接存取，需要占用较大内存空间

单级索引分配

每个文件分配一个索引快，存放分配给该文件的所有盘块的盘块号。所以表存放的磁盘块称为索引块，文件数据存放的磁盘称为数据块。
但当文件很大时，索引块会很大。读写速度相对不快。

两级索引结构

|主索引||第二级索引||磁盘空间|
第一级索引表存放索引表块块号，第二级索引表登记的是磁盘空间的内容。
先将一级索引表调入内存，查询目标所在的二级索引，再由二级索引找到目标。访问目标数据块需要三次磁盘IO

混合索引结构

将直接结构和索引结构结合。

1. 直接地址：提高文件检索速度，在索引节点中设置10个直接地址项addr(11)。每项中存放该文件数据的盘块的盘块号。
2. 一次间接地址：再利用索引节点地址项来提供一次间接地址。实质是一级索引分配方式。系统将分配给我呢间的多个盘块号计入其中。在一次间址块存放k个盘块号，因此允许文件长达4MB
3. 多次间接地址：文件长度大于`4MB+40KB`时，还要采用二次间址分配。此时用addr(11)提供二次间址，实质是两级所索引分配方式，最大文件长度4GB（`4KB/4B * 4KB/4B * 4KB = 4GB`）。采用addr(12)作为三次间接地址，文件最大长度可达4TB。访问磁盘次数=间址级别+1

磁盘空闲空间管理

1. 空闲表法：把每一个块的空闲空间列入空闲表中
2. 空闲链表法：把空闲磁盘块拉成一个单链表（空闲盘块链，空闲盘区链）
3. 位示图：行列中的每个交叉点表示第一个比特，1表示被分配，0表示未分配
    1. 分配：
        1. 顺序扫描位示图，找出一个或一组为0的二进制位
        2. 将该二进制位转换成与之相应的盘块号。设其位于位示图的i行j列，盘块号公式：`b = n(i-1)+j`，n表示每行位数
        3. 修改位示图，`map[i,j] = 1`
    2. 回收：
        1. 根据行号列号，转换公式为：`j = (b-1)/n + 1`，`j = (b-1)%n + 1`
        2. 修改位示图，`map[i,j] = 0`

文件的共享与保护

现代OS必须提供文件共享手段，允许多用户/进程共享同一个文件，系统只保留该共享文件的一份副本。

基于索引节点的文件共享（硬链接方式）

给每个节点提出一个硬连接，在索引中设置一个count值，若count==x表示x个用户在共享该文件。
只要count>0就表示还有别的用户使用该文件，暂时不能把文件数据删除，否则导致指针悬空；当count==0系统负责删除该文件。

基于符号链接的文件共享

软链接，例如快捷方式。只有文件主才有只想索引节点的指针，其它用户只拥有一个指向该文件的链接。

文件保护

影响文件安全性的主要因素：人为因素，系统因素（系统异常造成数据损坏或丢失，或磁盘出现故障），自然因素（随着时间推移，硬盘消磁）

文件保护：口令保护，加密保护，访问控制。
保护域机制：进程只能在保护域内执行操作，且只能访问具有访问权限的对象。
在每个文件的FCB或索引节点中增加一个访问控制表ACL，记录了各用户可对该文件执行的操作。

磁盘结构

读写磁头在轴心上的磁盘扫描，磁盘分为各个扇区（扇形）和磁道（同心圆）。根据柱面号移动磁臂，让磁头指向指定柱面；集火指定盘面对应的磁头；磁盘旋转的过程中，指定扇区会从磁头下划过。

磁盘地址 = 柱面(磁道)号-盘面号-扇区号；
柱面号 = 簇号/每个柱面的簇数；
盘面号 = (簇号%每个柱面的簇数)/每个磁道的簇数；
扇区号 = 扇区地址%每个磁道的扇区数。

磁盘访问时间

- 寻道时间Ts = m*n+s
  - 磁头移动到指定磁道上的时间
- 旋转延迟时间Tr
  - 指定扇区移到磁头下面的时间
- 传输时间Tt = b/(rN)
  - 数据从磁盘读出或向磁盘写入数据所经历的时间
- 平均读取时间 = 1/r
- 总时间Ta = Ts+1/2r+b/rN

对于7200r/min、平均寻道8ms、每个磁道1000个扇区的磁盘，访问一个扇区的平均存取时间：寻道时间8ms + 旋转延迟`60*1000/7200*(1/2)` + 读写延迟`60*1000/7200*(1/1000)`

磁盘寻道算法

多个进程访问磁盘时，磁盘调度应使磁盘平均寻道时间最短。

- 先来先服务FCFS
  - 根据进程访问磁盘的先后次序进行进行调度。公平、简单，进程请求都能依次得到处理。适合请求磁盘IO进程数较少程序
- 最短寻道时间优先SSTF
  - 算法选择要求访问的磁道与当前磁头所在磁道距离最近的进程，使本次操作寻道时间最短。但可能导致某些进程发生饥饿
- SCAN算法（电梯调度算法）
  - 不仅考虑欲访问磁道与当前磁道的距离，*更考虑*磁头的当前移动方向。算法能获得较好的寻道性能，又能防止进程饥饿，广泛应用于大中小型机和网络中的磁盘调度。但对某些磁道不公平
- CSCAN循环扫描算法
  - 不仅考虑欲访问磁道与当前磁道的距离，*更优先考虑*磁头当前移动方向。规定只有*磁头朝某个特定方向移动*时才处理磁道访问请求，而返回时直接快速移动到起始端而不处理任何请求
- N-Step-SCAN算法
  - 上述算法容易“磁臂粘着”（某几个进程对某一磁道有较高访问频率）
  - N-Step-SCAN算法将磁盘请求队列分为若干长度为N的子队列，磁盘调度将按FCFS算法一次处理每个子序列，每处理一个队列时又是按照SCAN算法，对一个队列处理完后，再处理其它队列。如果处理过程中出现新的IO请求，则将其放入其它队列，避免粘着

- FSCAN算法
    - 将N-Step简化，只将磁盘请求队列分为两个子序列：由当前所有请求磁盘IO的进程形成的队列，和新出现的请求磁盘IO的进程队列

磁盘管理

- 提高磁盘IO速度的途径：
    - 磁盘高速缓存：内存中为磁盘设置缓冲区
    - 提前读：采用顺序访问时，可以预先将磁盘数据先读入高速缓存
    - 延迟写：应写回磁盘的缓冲数据先挂在空闲缓冲区队尾
    - 优化物理块分布：同一文件的盘块尽量处于同一磁道或相邻磁道
    - 虚拟盘：通过内存空间模拟磁盘
    - 廉价磁盘冗余阵列RAID：通过资源重复解决系统瓶颈问题
- 提高磁盘可靠性的途径
    - 磁盘容错技术：增加冗余磁盘设备
    - 后背数据：利用后备系统暂存不需要的还有用的数据
- 磁盘低级格式化
    - 将整个磁盘划分为柱面和磁道，再将磁道划分扇区（将整个磁盘重新划分各区域）。步骤如下：
        1. 会将扇区清零和重写校验值，并尝试修复坏道；
        2. 对扇区标识信息重写；
        3. 对扇区进行读写检查，并尝试替换缺陷扇区；
        4. 对所有物理扇区重新编号；
        5. 写磁道伺服信息；
        6. 写状态参数，并修改特定参数。
    - 一般分为快速格式化、高级格式化、低级格式化。低级格式化在特殊情况才使用
- 磁盘逻辑格式化（高级格式化）
    - 根据一定分区格式对磁盘进行标记，生成引导区信息、初始化空间分配表、标注逻辑坏道、校验数据等。只对磁盘进行寻常的读写操作。

文件系统挂载

磁盘格式化后，磁盘分区要能够使用，必须挂载。在挂载某个分区前要先建立一个挂载点。

虚拟文件系统VFS

用于支持大量的文件管理系统和文件结构。VFS向用户提供一个简单统一的文件系统接口，定义了一个能代表任何文件系统的通用特征和行为的通用文件模型。

VFS认为文件是计算机大容量存储器上的对象。文件可被创建、读写、删除，任何文件系统都要一个映射模块，以便将实际文件系统的特征转换为虚拟文件系统所期望的特征。
VFS等同于向用户隐藏内核空间的细节。

VFS是用C语言实现的面向对象的方案，每个对象都包含数据和函数指针。
包含的主要对象：超级块对象，索引节点对象，目录项对象，文件对象

- 超级块：一个超级块对应一个文件系统，保存了该文件系统的类型、大小、状态
- 索引节点inode：保存实际的数据信息（元数据）。创建一个文件时就给文件分配了一个inode。一个inode只对应一个实际文件，inode最大数量等同于文件数量
- 目录项：描述文件的逻辑属性，只存在于内存，相当于存在内存的目录项缓存

- 文件对象：描述进程已经打开的文件

固态硬盘SSD

以半导体闪存为介质的存储设备。主要部件为控制器和存储芯片，硬件包括主控、闪存、缓存（可选）、PCB、接口。
基本存储单元分为：SLC、MLC、TLC。

SLC的存储单元里存储1bit数据，高于4v表示0（已编程），低于4v表示1（已擦除）。
MLC存储2bit，分为00、01、10、11。与SLC采用相同电压，但阈值被分为四份。
TLC存储3bit。电压阈值分的更细致。

动态磨损：
主控会选择较新闪存颗粒进行擦除写入。算法简单粗暴，主控处理压力小，占据资源小。但精细化程度不够导致无法全面覆盖和实现所有颗粒的磨损均衡。

静态磨损：
执行擦除写入时，优先把长久不用的冷数据从较新的闪存颗粒提出并放入老颗粒，并将新写入的数据放在较新颗粒，实现均衡化。颗粒寿命高，但算法复杂、主控压力大。

补充

- 一个文件被用户进程首次打开的过程中，操作系统需要将文件控制块FCB读到内存中
- 文件被删除时，OS不可能执行的是删除此文件所在目录
- 索引节点个数与文件个数一一对应
- 索引节点所占位数可以反映最多文件个数（例如索引节点占2B，则有16位可以表示索引节点，得到最多65536个文件）
- 随机访问与动态扩张

分配方式	随机访问	动态扩张
连续分配	√	×
链接分配	×	√
索引分配	√	√

- F1的count=1，建立F1软链接文件F2，建立硬链接F3，删除F1后，F2、F3的引用计数值为1、1

IO设备管理

IO是输入输出，IO设备可以将数据输入到计算机、或从计算机输出数据到IO设备

IO设备分类

- 按传输速率分类
  - 低速设备：几字节到几百字节每秒，鼠标、键盘、语音输入输出等
  - 中速设备：千个到数万个字节每秒，打印机等
  - 高速设备：数百千到数十兆字节每秒，光盘、磁带等
- 按信息交换单位分类
  - 块设备：用于存储信息且信息存取总以数据块为单位，典型的块设备是磁盘
  - 字符设备：用于数据输入输出，基本单位是字符
- 按使用方式分类
  - 独占设备：一段时间内只允许一个用户/进程访问的设备，例如打印机这一临街资源必须互斥访问的设备
  - 共享设备：一段时间内允许多个进程同时访问，例如磁盘为可寻址、可随机访问的设备
  - 虚拟设备：通过虚拟技术会将一台独占设备变为若干台逻辑设备

IO控制器

分为机械部件和电子部件。其中电子部件就是常说的IO控制器（接口、控制接口、模块）

IO控制器功能

- 数据缓冲：主存和CPU寄存器的速度很快，外设速度较低。在设备控制器汇总引入数据缓冲器用于缓存数据
- 错误和就绪检测：提供错误和就绪时需检测逻辑，并将结果保存在状态寄存器供CPU查用
- 控制和定时：接收和识别CPU或通道发来的控制信息和定时信号，根据相应信息和信号想外设发送控制信号
- 数据格式的转换：提供数据格式转换部件，通过外部接口得到的数据转换为内部接口需要的格式
- 当链接多台设备时，设别地址识别

IO端口

CPU能够访问的各类寄存器称为IO端口，驱动程序通过IO端口控制外设进行IO。对IO端口读写就是向IO设备送出命令或从设备读状态或读写数据。

- 控制端口（命令端口）：存放CPU通过out指令向外发送的控制命令
- 状态端口：CPU通过in指令读取状态寄存器了解外设和设备控制器的状态
- 数据端口：访问数据缓冲寄存器进行数据的IO

一个IO控制器会有多个端口地址，IO端口编号后才能被CPU访问，IO设备的寻址方式就是IO端口的编号方式：

1. 独立编址方式（IO映射方式）：外设与主存单元的地址分别单独编址，外设端口不占用主存空间。但CPU要专门设置输入IO指令访问端口，增加了控制复杂性
2. 同一编址方式：外设端口和主存单元的地址统一编址，区分靠不同的地址码，不需要给IO端口额外的编址。但占用了存储器地址空间，执行速度较慢

IO控制方式

CPU与IO设别实现数据交换的方式。CPU和外设之间交换数据，实质上是通过IO端口进行。

- 程序查询方式：完全通过CPU执行IO程序控制主机和外围设备间的信息交换
  - 需要查询指令（查询设备状态）、传送指令（设备就绪时执行数据交换）、转移指令（数据未就绪时转向查询指令继续查询）
  - 不断查询IO设备情况，终止源程序执行，完全处在串行工作状态
- 程序中断方式：允许IO设备主动报告自己的状态，不需要CPU不断查询
- DMA方式（直接存储器存取）：在外围设备和主存间开辟直接的数据传送通路
  - 需要寄存器：命令/状态寄存器CR、内存地址寄存器MAR、数据寄存器DR、数据计数器DC
  - 需要DMA中断机构，当一个数据块传输完成后触发中断机构向CPU提出中断请求
  - 由于是异步方式工作，数据块传输结束的时机必须由DMA中断机构以中断方式向CPU告知
- IO通道控制方式：一种特殊的具有执行IO指令能力并通过执行通道IO程序来控制IO操作的处理机
  - 指令类型单一，通道硬件比较简单，其执行的命令主要局限于与IO相关的操作操作有关的指令
  - 通道与CPU共享内存
  - 主要是工作站等专业设备使用
- IO处理机：可以不通过CPU来独立控制IO设备

IO软件

IO软件的设计目的和原则：与具体设备无关，统一命名，对错误处理，缓冲技术等

IO软件层次结构

（驱动程序直接将请求信息发给硬件，硬件发送应答给中断处理程序，再发给驱动程序）：

- 用户进程
- ↓IO请求 || ↑IO应答
- ↓用户层软件↑ ← 交互接口，调用函数库
- ↓设备独立性软件↑ ← 设备的命名、保护、分配、回收
- ↓设备驱动程序↑（可重入） ← IO命令转为具体要求，DC和软件的接口
- |↓| 中断处理程序↑ ← IO操作结束被唤醒，处理中断
- 硬件↑ ← 执行IO操作

层次	作用
用户层软件	实现与用户交互的接口，用户可直接调用再用户层提供的与IO相关操作的库函数，对设备进行操作
设备独立软件	（设备无关软件）实现设备独立性，让应用程序独立于具体物理设备，使用逻辑设备名来请求某类设别（再系统实际执行时还需使用跟物理设备名称）
设备驱动程序	IO进程和设别控制器之间的通信程序。接收上层软件发来的抽象IO要求，转换为具体要求后发给设备控制器启动设备并执行。为每一类设备设置一个进程，专门用于执行这类设备的IO操作

层次	作用
中断处理程序	进行进程上下文切换，对处理中断信号源进行测试，读取设备状态和修改进程状态

IO缓冲处理

CPU实现与IO设备进行数据交换的方式。CPU和外设实质上通过IO端口进行交换数据：`CPU <---> IO接口/端口 <---> IO设备`。

IO缓冲的目的：减少CPU被中断的频率，放宽CPU的中断响应时间限制，提高CPU和IO设备间的并行性，缓解CPU和IO设备的速度矛盾：
`CPU <---> 缓冲区 <---> IO`。
一般多使用内存作为缓冲区。

单缓冲：只设置一个缓冲区，用户之间发数据只能以"半双工"形式；
双缓冲：设置两个缓冲区，可以同时输入或输出，也可以使得用户间可以"全双工"形式。

T：IO设备将数据输入单缓冲的时间；
M：OS将缓冲数据传至内存用户区的时间；
C：CPU对数据的处理时间；
`总处理时间 = M + max(C+M,T)`。 //对于多块设备需要额外加上C；
若C+M = T，则双缓冲区构成二集流水线结构

设备的分配、回收

- 分配策略以及考虑因素
  - 静态分配：一次性分配进程所需全部设备
  - 动态分配：进程执行过程中根据执行需要按策略分配
  - 设备分配算法：
    - 先请求先分配
    - 优先级高者优先
  - 设备固有属性：不同属性采取不同的分配策略
    - 独占设备：得到设备的进程独占，直到完成或其他原因释放设备
    - 共享设备：分配给多个进程，并合理分配各进程的访问顺序
    - 虚拟设备：共享设备，可将其同时分配多个进程
  - 设备回收：释放设备、设备控制器、通道；修改相应数据结构
  - 安全性：
    - 安全分配：进程发出IO请求后进入阻塞，串行工作
    - 不安全分配：进程发出IO请求后仍可运行，并行工作
- 逻辑设备名到物理设备名的映射

设备分配中的数据结构：

- 设备控制表DCT：系统为每个设备配一个DCT，记录状态
- 控制器控制表COCT：系统为每个控制器配一个COCT
- 通道控制表CHCT：系统为每个通道配一个CHCT
- 系统设备表SDT：记录系统全部设备情况的表，每个设备占一个表目

- 逻辑设备表LUT：存放逻辑设备名、物理设备名、设备驱动程序入口地址的表，用于完成逻辑设备到物理设备的转换

假脱机SPOOLing

缓和CPU和IO设备的高低速矛盾而引入。当系统引入多道程序技术后，完全可以利用其中两道程序模拟脱机输入和输出的外围控制机功能，实现低速设备和高速设备的脱机输入、输出。

SPOOLing技术把一台物理设备虚拟成逻辑上的多台设备，将独占式设备改为共享设备。提高了IO速度，将独占设备改造为共享设备，实现了虚拟设备功能。

补充

- 区分硬件和识别设备的代号称为设备的绝对号（硬件角度）
- 对多个同一类硬件（例如多台打印机）只需提供一个设备驱动程序
- 将占10个硬盘块的问津逐个读入主存缓冲区并送入用户区分析，磁盘块读入缓冲区耗时100us，缓冲区数据传至用户区50us，CPU分析数据50us
    - 单缓冲区结构：10*(100+50) + 50 = 1500
    - 双缓冲区结构：10*(100) + 100 = 1100
- SPOOLing提高了单机资源利用率