操作系统

《操作系统教程 第4版》孙钟秀 高等教育出版社ISBN 978-7-04-023221-9

## 操作系统概述

1. 讨论操作系统的4种观点：1）用户接口与公共服务程序：2）程序的控制者和协调者；3）扩展机与虚拟机；4）资源管理者与控制者；

操作系统资源管理功能主要有：处理器管理、存储器管理、设备管理、文件管理、网络与通信管理；为了实现资源管理，主要使用资源复用，虚化和抽象技术；通过进程抽象，虚存抽象和文件抽象来创建一个虚拟机；

1. 操作系统是一个大型复杂的并发系统，并发性、共享性、随机性（异步性）是它的重要特征；
2. 通道技术的引进和中断技术的发展，导致了操作系统由单道作业处理进入了多道程序设计系统阶段；实现多道程序设计必须妥善处理：1）存储保护与程序浮动；2）处理机的管理与分配；3）资源的管理与调度；

这一阶段，操作系统沿着三条主线发展：多道批处理系统、分时交互系统和实时处理系统；

多道批处理系统着眼于让处理器和外围设备同时保持忙碌，提高作业的吞吐率和整个系统的效率；其关键机制是：在响应一个作业的处理结束信号时，处理器将在主存中驻留的不同作业间切换；

分时交互系统的主要设计目标是为用户提供方便的程序开发、调试环境和要快速响应交互式用户的命令请求，但又支持多用户同时工作，以降低系统的成本；由于用户键盘操作速度较慢，快速响应用户请求和同时支持多用户这两个目标是可以共存的；其关键机制是：采用时间片轮转法，让处理器在多个交互式用户间多路复用；

与分时系统相比，实时事务系统常局限于一个或几个应用，例如，数据库的查询和修改应用或生产过程控制实时应用，有响应时间的限制；其关键机制是：事件驱动机制，当系统接受来自外部的事件后，快速分析这些事件，驱动实时任务在规定的响应时间完成相应处理和控制；

1. 操作系统的基本类型有三种：批处理系统、分时系统和实时系统；凡具备全部或兼有两者功能的系统称通用操作系统；随着硬件技术的发展和应用深入的需要，新发展和形成的操作系统有：微机操作系统、网络操作系统、分布式操作系统和嵌入式操作系统；
2. 操作系统的结构：单内核（整体式结构、层次式结构）、微内核结构；内核是由中断驱动的，即只有在中断或异常发生后才会引出内核工作，处理完成后内核就会退出；内核与裸机为每个进程形成了一台没有中断的虚拟机，进程独占虚拟的CPU，内存和外设，不需要处理中断；

## 处理器管理

1. 操作系统是由中断激活的，中断装置识别并响应中断事件，通过交换PSW，让中断处理程序占有处理器；在处理完该中断事件后，通常会改变一些进程的状态，引起相应进程队列的调整，然后转向低级调度执行调度工作；中断能改变处理器执行的顺序，是操作系统实现并发性的硬件基础之一；中断也是用户态向核心态转换的唯一途径；
2. 为了屏蔽中断的影响，操作系统较早地引入了一个由并发执行的顺序程序组成的概念模型—进程，通过中断方式进程可以在实处理器上交替执行，每个进程可以认为都运行在自己的一个虚拟处理器上；
3. 进程是程序的执行，是一个动态的概念；进程映像是某时刻进程的内容与状态的集合，包控：进程控制块（PCB），程序，数据（私有栈等），核心栈；进程上下文是处理器的一个执行环境，包括：用户级上下文（程序，数据，私有栈等）、系统级上下文（PCB，页表，核心栈等）、寄存器上下文（PSW，PC，SP，控制寄存器，通用寄存器等）；
4. 进程的三态模型：1）运行态；2）就绪态；3）等待态（又称阻塞或睡眠）；可能的状态转换及原因：
   1. 运行→等待：运行进程等待使用某种资源，或某事件发生，如设备传输或键盘输入；
   2. 等待→就绪：所需资源得到满足或某事件已完成；
   3. 运行→就绪：运行时间片到时，或出现更高优先级的进程，当前进程被迫让出CPU；
   4. 就绪→运行：调度进程选中当前进程执行；

进程在运行过程中执行系统调用，或发生中断时，操作系统获得控制权，当发现当前时间片耗尽，或执行一个**阻塞型**系统调用（如进程创建，退出等），就会导致进程切换和调度；

1. 操作系统中引入线程后，进程涉及到资源所有权及其保护，线程是调度的独立单位，涉及到程序的执行；线程实现方法：用户级、系统级和混合式；
2. 处理器调度有抢占式和非抢占式两种，按层次分，有高级调度，中级调度和低级调度；

**高级**调度又称作业调度，用来决定满足资源需求的后备作业选中进入主存去运行，控制多道程序的道数；

* 1. 先来先服务，FCFS，用队列实现；
  2. 最短作业优先，SJF，非抢占式，平均周转时间最短，但长作业可能发生“饥饿”；
  3. 优先级调度，既可用于抢占式，也可用于非抢占式；

静态优先级，定级方法：按进程类型（系统/用户，前台/后台），资源需求，用户类型等；

动态优先级，定级方法：已占用CPU时间长短，等待时间长短；

相同优先级的则可按FCFS或SJF调度，形成多级队列；

* 1. 高响应比优先：响应比=作业响应时间/估计运行时间=（等待时间+估计运行时间）/估计运行时间；有利于短作业，也考虑到了等待时间，不会导致长作业“饥饿”；

**中级**调度主要是根据主存大小决定进程数目，并决定与辅存的对换（Swap）；

**低级**调度又称进程/线程调度，决定哪一个进程或线程占有处理器运行，是各类操作系统必备的功能和最为核心的部分，执行十分频繁，调度策略的优劣直接影响整个系统的性能；

1. 先来先服务；
2. 最短进程优先；
3. 优先级调度；
4. 轮转调度，RR，使用时钟中断划分（均等或可变长）时间片；抢占式；时间片短则进程切换开销比率大，时间片长则接近FCFS的效果；
5. 多级反馈队列，MLFQ，又称反馈循环队列，是时间片轮转和优先级调度的结合；每个优先级对应一个就绪队列；高优先级分配的时间片**短**；通过对等待时间反馈来提高低优先级队列中进程的优先级，防止“饥饿”；
6. 彩票调度，彩票代表了资源的份额，容易实现按比例分配资源；
7. 实时系统调度算法：单比率调度、限期调度、最少裕度调度；
8. 多处理器调度算法：负载共享调度、群调度、专用处理机调度、动态调度；
9. 衡量调度算法优劣的指标有：资源利用率、作业吞吐率、公平性（不会出现饥饿）、响应时间（分时/交互式，用户提交命令到得到响应）、周转时间（批处理，从提交作业到作业完成）等；

## 同步、通信与死锁

1. 进程可并发执行的条件：1），且，保证一个进程在读取存储器时，其中的数据不发生变化；2），保证写操作不会覆盖丢失；
2. 并发进程之间存在两种基本关系：竞争和协作；

死锁指的是两个或更多的事务同时处于等待状态，每个事务都在等待其他的事务释放锁使其可继续执行；事务故障是事务没有按预期的要求成功完成，产生的原因是逻辑错误或系统错误。

并发进程可能需要竞争使用资源，资源竞争可能导致死锁或“饥饿”；引入了**临界区**（并发进程中与共享变量有关的**程序段**）以解决进程互斥问题；临界区使用准则：**空闲让进，忙则等待，有限等待，让权等待**；

Perterson算法：

|  |  |
| --- | --- |
| // Perterson算法是一种谦让算法，多个进程共享一组标志inside[]，每个进程都可以读取其它进程的标志，  // 但只设置自己的标志，表示自己希望进入临界区（访问共享数据）  bool inside[2];  inside[0]=false;  inside[1]=false;  int turn; //共享变量，指示哪个进程可以进入临界区，每个进程都把turn设置为另外一个进程，以示谦让 | |
| void ProcessA(){  inside[0]=true;  turn=1;  while(inside[1]&&turn==1);  //A的临界区代码  inside[0]=false;  } | void ProcessB(){  inside[1]=true; //B希望进入临界区  turn=0; //谦让一下，先让A进入  while(inside[0]&&turn==0); //若A想进且**没轮到B**，则循环，等待  //B的临界区代码 //结束循环等待，可以执行临界区  inside[1]=false; //退出临界区  } |

Perterson算法是一种解决互斥的软件方法，其中的while语句相当于一个自旋锁spinlock，此外还有硬件的方法，如关中断（单CPU），锁总线（多CPU），专用原子指令（XCHG）等；spinlock虽然让CPU处于忙等状态，浪费了CPU时间，但与信号量方式相比，不会强制阻塞当前线程从而导致进程切换和调度，适合细粒度的场合；

**同步**是解决进程间协作关系的手段；进程同步的主要任务是使并发协作的进程之间能有效地共享资源和相互协作，协调各进程的进度，从而使进程的执行过程具有可再现性和执行结果的唯一性；

**互斥**是一种特殊的进程同步关系；同类进程一般是互斥关系，不同类进程一般是同步关系；

进程间的低级通信机制一般只是传递信号，主要有：原子操作，锁机制，信号（Signal），信号量（Semaphore）和PV操作，管程等；使用低级通信机制解决许多经典同步问题的实例，如生产者与消费者问题、读者与写者问题、5个哲学家吃通心面问题、睡觉的理发师问题，这些问题是计算机操作系统中并发进程相互制约和内在关系的一种抽象；从理论上来说，各种同步机制都是等价的，每种机制都可以用另一种来实现，但在实际应用中，信号量，消息传递和共享主存方式应用较多；

1. 信号量Semaphore是一种特殊的结构变量，代表了某个资源；

Semaphore包括一个有符号整数值cnt，当cnt为正，表示可用资源数量，当cnt为负，表示等待（阻塞）该资源的进程数量；为此，还要有一个队列queue来记录这些等待进程；

Semaphore只允许两种操作，P（检测），V（增加），它们都是原子性的，保证对信号量本身的操作不会引起冲突；

1. P操作： cnt--，表示占用了一个资源，如果cnt<0，则阻塞当前进程，并将其加入等待队列queue中，否则不特别处理，当前进程继续执行；
2. V操作： cnt++，表示释放了一个资源，如果cnt<=0，则从等待队列queue中唤醒一个进程，将其转入就绪态，否则不特别处理，当前进程继续执行；
3. 信号量解决互斥问题：一组同类进程，初始cnt=1，进程执行P进入临界区，退出后执行V操作释放信号量；

|  |  |
| --- | --- |
| semaphore mutex;  mutex =1; | // 信号量是由多个进程共享的  // 设置为1即可实现进程互斥 |
| void ProcessA(){  P(mutex);  //A的临界区代码  V(mutex);  } | void ProcessB(){  P(mutex); //mutex-1，若B先执行P，则阻塞A，否则阻塞  //B的临界区代码 //B自己，从而实现互斥操作  V(mutex); //退出临界区  } |

1. 信号量解决同步问题：一组不同类进程，初始cnt为0或某个正整数，在**不同的进程**里分别执行P，V操作，实现通信，注意**P，V一定是成对的**；

单缓冲，单个生产者和单个消费者问题，需要信号量实现进程同步，表明缓冲区是满还是空；

|  |  |  |
| --- | --- | --- |
| semaphore empty,full;  empty =1;  full =0; | //empty指空闲缓冲区数量，full指已填充缓冲的数量  //初始状态缓冲区是空闲的 | |
| void Producer(){  while(true){  getData();  P(empty); //确认缓冲区有空闲  fillBuffer();  V(full); //通知消费者  }  } | | void Consumer(){  while(true){  P(full); //确认缓冲区不是空的  clearBuffer();  V(empty); //通知生产者  processData();  }  } |

大小为k的缓冲，多个生产者和多个消费者问题，需要信号量实现进程同步，表明缓冲区是满还是空；此外还要有一个互斥量防止生产者或消费者同时访问缓冲区；

|  |  |  |
| --- | --- | --- |
| semaphore empty,full,mutex;  empty =k; full =0;  mutex =1; | //empty指空闲缓冲区数量，full指已填充缓冲的数量  //mutex实现互斥访问缓冲区 | |
| void Producer(){  while(true){  getData();  P(empty); //确认缓冲区有空闲  P(mutex); //特别注意两者顺序，防止死锁  fillBuffer();  V(mutex);  V(full); //通知消费者  }  } | | void Consumer(){  while(true){  P(full); //确认缓冲区不是空的  P(mutex);  clearBuffer();  V(mutex);  V(empty); //通知生产者  processData();  }  } |

理发师问题：1位理发师，1把理发椅，N把供顾客等候的椅子；如果没有顾客，理发师就在理发椅上睡觉，有顾客到了，会唤醒理发师理发；如果理发师正忙，有顾客到了则坐在椅子上等，如果没有空椅子则离开；

|  |  |  |
| --- | --- | --- |
| const int RESTING\_CHAIR\_CNT=N;  int waitingCustomerCnt=0;  semaphore barber, customer, mutex;  barber =0; customer =0;  mutex =1; | //常量，供顾客等候的椅子的数量；  //共享变量，正在等待的顾客的数量，需用mutex实现互斥访问；  //barber指示理发师状态，customer指示顾客状态 | |
| void Barber (){  while(true){  P(customer); //确认有顾客，则开始理发  P(mutex); //特别注意两者顺序，防止死锁  waitingCustomerCnt --;  V(mutex);  cutHair();  V(baber); //服务完一位顾客，通知其它  }  } | | void Customer (){  P(mutex);  if(waitingCustomerCnt<RESTING\_CHAIR\_CNT){  waitingCustomerCnt ++; **V(mutex);**  V(customer); //通知理发师有顾客来了  P(baber); //确认理发师有空，等待理发  } else {  **V(mutex);** //等候的顾客太多，离开  }  } |

1. 管程（Monitor）是把共享变量及其操作集中在一起统一控制的模块，提供了一种互斥机制；
2. 进程间的高级通信（IPC）主要用来传递数据，有共享内存；消息传递（直接通信，信箱）；管道（共享文件）；不同进程间的消息传递，由于内存地址空间的隔离，需通过操作系统才能完成；消息传递的同步方式通常采用非阻塞型Send和阻塞型Receive；
3. 死锁：无论是相互通信的进程或是共享资源的进程，都可能因推进顺序不当或资源分配不妥而造成死锁；死锁是系统中一组并发进程因等待其它进程占有的资源，而永远不能向前推进的僵化状态；
4. 产生死锁的4个必要条件：**互斥条件、占有并等待条件、不剥夺条件和循环等待条件**；只要破坏这4个必要条件之一，就可以防止死锁；
5. 解决死锁问题的三种策略和方法：预防、避免、检测和解除；

死锁的预防是系统**预先确定**一些资源分配策略，进程按此规定来申请和使用资源，保证死锁的某一个必要条件不会满足，使得系统不发生死锁，缺点是资源利用率低，或对资源使用的限制过严；

死锁的避免涉及到**动态地分析和检测**新的资源请求和资源的分配情况，以确保系统始终处于安全状态，对资源的使用放宽了条件；**银行家行算法**是著名的死锁避免算法，但该算法要求预先获得有关信息，很少有进程能够在运行前就知道其所需的最大资源量，而且系统中进程数不是固定的，往往在不断地变化，所以该算法缺乏实用价值；

若有个资源，个进程共享使用，每个进程需要的**最大**资源数为，由银行家算法，时系统是安全的；若，则；若，则；

死锁的检测和解除算法中，操作系统总是同意资源申请，对资源的分配不加任何限制，也允许系统发生死锁，但必须建立一个检测机制，选择检测算法（如进程资源分配图、传递闭包等），周期性地检测是否发生了死锁，如果发生了，把它检测出来，再采取措施去解除死锁；虽然死锁检测对资源的使用不加任何限制，但开销比较大；

## 存储管理★

1. 存储管理的基本功能有：存储分配、地址转换（抽象和映射）、存储保护（隔离）和共享、存储扩充（虚存）；

存储分配是给多道运行的作业分配主存空间；

地址转换是为用户程序提供一个单一的，足够大的，连续的抽象逻辑地址空间，把逻辑地址空间中的用户程序通过静态重定位或动态重定位映射到分配的物理地址空间中，方便用户程序的编译和执行；

存储保护指各道程序只能访问自己的存储区域，而不能互相干扰，以免程序受到有意或无意的破坏；存储共享指主存中的某些程序和数据可供不同用户进程同时使用；

存储扩充是将辅存抽象为虚拟主存，允许用户进程的逻辑地址空间大于实际的物理内存容量；

1. 内存分配需要对空闲的内存进行管理，可采用bitmap/数组或空闲链表的方法；

最简单的单道系统中，如果一个程序能装入主存，那么它将一直运行，直到结束才返回操作系统；

如果主存空间不足，可以通过**覆盖**或**交换**的技术解决；

多道程序操作系统最简单的存储管理技术是分区方式，又分固定分区和可变分区；可变分区的**分配算法**有：最先适应（空闲分区按地址排序，常用）、下次适应（从上次扫描结束处顺序查找最合适的空闲分区，可避免碎片集中在前端）、最佳适应（空闲分区按大小排序）、最坏适应、快速适应（常见大小分区单独管理，回收算法较复杂）；

伙伴系统是介于固定分区和可变分区之间的一种分配方案；

采用分区方式管理内存，每道程序总是要求占用主存的一个或几个连续存储区域，会产生许多碎片；如果主存空间不足，可以通过**拼接**（移动分区，合并碎片）、**覆盖**、**交换**技术解决；

现代计算机支持虚存，如请求分页式虚存管理，允许把一个进程的页面存放到若干不相邻接的主存页框中，消除了外碎片，但每个进程存在不多于一页的内碎片；

1. 虚拟存储器基于程序的局部性原理，不把一个用户进程的全部信息同时装入主存，而是仅将其中当前使用部分先装入主存储器，其余暂时不用的部分先存放在作为主存扩充的辅存中，待用到这些信息时，再由系统把它们装入到主存中，简而言之，它采用了自动的部分装入、部分对换、主存辅存独立编址但统一使用的技术，使得进程的虚拟地址空间可以远远大于物理内存容量，为用户编程提供了极大方便；通过虚拟存储器，所有的用户程序和数据都给出逻辑地址，在运行时由系统转换成物理地址，即动态地址转换；于是就允许一个进程位于主存的任何地址，它的位置还可以动态改变；

分页式虚存管理实现页面共享较复杂；

多级页表：减少页表本身占用的存储空间，增加了访存次数；

快表：页表的高速缓存，减少地址映射的访存次数，全相联存储；

1. 反置页表：主要是为了解决64位系统中页表过大的问题；传统的页表，以逻辑页号为序号（无需存储），页表内容为对应的物理页号，适应逻辑页号→物理页号单方向的转换，每个进程有自己独立的一张页表；而反置页表是整个系统只有一张页表，以物理页号为序号（无需存储），页表内容为逻辑页号，为区分不同进程，还需记录进程PID；

为实现逻辑页号→物理页号的转换，需根据PID和逻辑页号按内容查找整个反置页表，为提高效率，可为系统增加一个Hashtable，按PID和逻辑页号计算hash，然后得到物理页号；

反置页表中只包含已经调入内存的页面，并未包含未调入内存的页面，因而必须为每个进程建立一个外部页表，形式与传统页表一样，一般放在辅存，缺页时使用；

反置页表实现共享页面较复杂；

1. 段式虚存是为满足现代高级程序设计语言二维地址的要求而引进的，按用户应用中逻辑上有完整意义的内容进行分段；为了实现段式存储管理，需要建立**每个进程的段表**，记录用户逻辑段与主存中物理段的对应关系，段式存储管理主存的分配与回收跟可变分区方式类似，可以通过直接分配、移动分配或调出分配来完成；

如果在分页式存储管理的基础上实现分段式存储管理，就是段页式虚存管理；

1. 分页涉及以下的问题：

页面装入策略：决定页面何时被装入主存，有请求式和预调式2种；

页面清除策略：决定**修改过的**页面**何时**被回写到磁盘上，有请求式（被换出时）和预约式（进程自主决定）2种；

页面分配策略：根据进程生命周期中分给的页框数可否改变，分为固定分配和可变分配；

页面替换策略：当内存中没有空闲页面时，需将已调入的若干页面写入辅存，腾出空闲页；根据作用范围是整个系统还是局限于某个进程内，分为全局页面替换算法（不区分进程）和局部页面替换算法（某个进程）；

固定分配策略通常和局部替换策略配合使用；可变分配策略既可以配合全局替换策略，也可以采用局部替换策略；

1. 全局页面替换算法：
2. 最佳页面替换算法OPT，淘汰以后不再使用的页（向后看）；理论上最优，无法实现，用于评价其它算法；
3. 先进先出FIFO，淘汰最先进入的页面（亦即驻留时间最长的），可能出现Belady异常（增多可用页面**可能**反而使缺页率上升）；因为驻留时间长的页面很可能是频繁使用的页面，被替换掉后，可能还会用到，导致缺页；
4. 第二次机会算法SCR：FIFO的改进，利用引用位，如果引用位为0，说明没有使用过，将其替换掉；如果为1，说明使用过，不替换此页，并将引用位置为0，并移至队尾，再给一次机会；继续扫描，寻找可替换页；如果所有页面的引用位都为0，那么第二遍扫描就会把队首的页面替换掉，这相当于FIFO；
5. 时钟页面替换算法CLOCK：将第二次机会算法的队列改为**循环**队列，淘汰页面时，从指针当前指向的页面开始循环扫描队列，如果引用位为1，则置0并跳过此页；如果引用位为0，则将该页淘汰，并且指针推进一步；
6. 最近最少使用LRU：淘汰一段时间内最久未使用过的页面（向前看），是基于时间局部性原理；需建立页面队列，每使用一次页面就要调整队列的顺序，队首总是最近最少使用的页，从而被淘汰；模拟方法：a)淘汰引用位为0的页面，又称最近未使用页面替换NRU；b)计数器记录使用次数，淘汰最少的页，又称最不经常使用页面替换NFU；c)计时法，每次使用页面刷新绝对时间戳（值是递增的），淘汰时间戳最小的（即最近没有使用过的）页面；以上三种方法都要定时清零，而且多需增加硬件支持；
7. LRU性能较优，不会出现Belady现象；CLOCK算法性能接近LRU，且容易实现；
8. 抖动现象指因页面替换算法不佳，页面被淘汰不久后，需要被调入使用，不久之后又被调出，如此反复的现象；
9. 局部页面替换算法：
10. 局部最佳页面替换：理论分析用；
11. 工作集模型和替换算法：工作集指某一时间段内进程运行所需使用页面的集合，是程序局部性的近似表示，确定进程常驻集的大小，使得进程缺页率低，解决抖动现象，又不至于浪费内存；工作集大小随时间变化；持续跟踪页面变化较困难，选取合适的时间段（工作集窗口）也是一个难题；
12. 模拟工作集替换：a)老化算法，引用位+老化字段；b)引用位+时间戳；
13. 缺页频率替换PFF：若相邻两次缺页时间超过临界值t，则移除所有未使用过的页，缩小工作集，反之则增大；
14. 随机替换算法也不失为一种简单有效的方案；

## 设备管理

1. 设备和CPU之间的数据传送控制方式主要有4种：程序轮询方式、中断控制方式、DMA方式和通道方式；

程序询问方式和中断控制方式仅适用于配置少量外设的场合，前者浪费CPU时间，设备与CPU只能串行工作；后者虽改进了上述缺点，但中断次数太多，CPU累计用在处理中断上的时间很可观，并行操作的设备数量受中断处理速度的限制；

DMA和通道方式较好的解决了上述问题，它们均采用了设备与和主存直接交换数据的方法，仅当一块数据传送结束，这两种方式才发出中断信号请求CPU干预，把CPU从繁杂的IO事务中解放出来；它们的区别是：DMA要求CPU执行设备驱动程序启动设备，并做好传送数据的有关准备工作；通道则完全是一个相对独立的IO控制系统，当CPU发出IO启动命令后，它便接收控制权，完成全部IO操作；

1. IO软件分4层：**设备中断处理程序、设备驱动程序、与设备无关的IO软件、用户层IO软件**；

**设备中断处理程序**通常是设备驱动程序的组成部分，位于底层，主要工作有：分析中断类型，作出相应处理，检查和修改进程状态等，它的任务要尽量少，以提高性能；

**设备驱动程序**中包括了所有**与设备相关**的代码，它的工作是：把用户提交的逻辑IO请求转化为物理IO操作的启动和执行，如设备名转化为端口地址、逻辑记录转化为物理记录、逻辑操作转化为物理操作等；它对上层的软件屏蔽所有的硬件细节；

**与设备无关的IO软件**的基本功能是执行适用于所有设备的常用IO功能，并向用户层软件提供一个一致的接口，如设备命名、设备保护、缓冲管理、提供与设备无关的存储块尺寸、设备分配和状态跟踪、错误处理和报告等；

**用户层IO软件**包括在用户空间运行的IO库例程和SPOOLING程序；

1. 通道又称为IO处理机，通道技术实现了IO的独立性，及设备与CPU的并行性，大幅度提高系统整体性能；具有通道的计算机系统，IO程序设计涉及CPU执行IO指令，通道执行通道命令，以及CPU和通道之间的通信；

与通道有关的若干概念，包括：CPU的IO指令、通道的通道命令、通道程序、CCW，CAW和CSW（通道控制字，地址字和状态字）、IO主程序和通道程序的编写、通道的启动和IO操作的执行等；

1. 缓冲主要用于匹配设备和CPU的处理速度，还可以减少中断次数；IO的一个重要特点是使用缓冲区；常用的缓冲技术有：单缓冲、双缓冲、多缓冲（循环缓冲，缓冲池）；缓冲区是由IO实用程序处理的，**而不是**由应用进程控制；

高速缓存Cache与缓冲Buffer的区别：Cache内的数据是低速存储设备上数据的一部分，相当于备份，而Buffer内的数据一般是独有的，不一定有备份；若Cache缺失，可以直接访问低速存储，但Buffer是数据的必经之路；

1. 为了提高磁盘IO的性能，采用了**磁盘驱动调度**和**磁盘缓冲区**；此外还有优化数据块分布方法；

常用磁盘驱动调度算法有：先来先服务FCFS、电梯调度（来回扫描，没有就掉头）、最短查找时间优先SSTF（可能饥饿）、扫描SCAN（来回扫描，一定到头，可能饥饿）、分步扫描（分组处理，防止独占）、单向扫描（单向扫描，到头复位）等调度算法；

磁盘缓冲区通常在主存开辟，作为磁盘块在磁盘与其余主存之间的高速缓冲；由于程序局部性原理，磁盘缓冲区的使用可大幅度减少磁盘与主存之间IO传送的次数；**提前读，延迟写，RAM盘**；

1. 独立磁盘冗余阵列RAID采用一组较小容量的、独立的、可并行工作的磁盘驱动器组成阵列来代替单一的大容量磁盘，再加上冗余技术，数据能用多种方式组织，分布存储；能并行对多个磁盘读写，改进了IO性能和系统可靠性；
2. RAID 0：并行交叉存取（条带），速度快，但无校验，只要有一个磁盘损坏，便会造成数据丢失，可靠性差，故较少使用；
3. RAID 1：磁盘镜像，可靠性高，但磁盘容量的利用率只有50%，读写速度相当于一块磁盘；
4. RAID 1+0：或称RAID 10，用4块硬盘，先做镜像，再做条带，结合了RAID 0和RAID 1的优点；
5. RAID 2：RAID 0的改进，增加了海明码校验，技术复杂，很少使用；
6. RAID 3：与RAID 2相似，但使用奇偶校验码，并用单独的一块磁盘来存储校验码，这块校验盘可能成为瓶颈；
7. RAID 4：与RAID3相似，但以块为单位校验；
8. RAID 5：校验码分布在各磁盘上；速度和可靠性都有提高，且得到较好的平衡，较常用；
9. RAID 6：双重校验措施，可靠性最高，写入速度差，较少应用；
10. SPOOLING系统能把一个物理设备虚拟化成多个虚拟（逻辑）设备，用共享设备来模拟独享设备；在中断和通道的支持下，操作系统采用多道程序技术，合理分配和调度各种资源，实现联机的外围设备同时操作；

SPOOLING系统主要由预输入、井管理和缓输出组成，已被用于打印控制和电子邮件收发等许多场合；

缓存在内存，井在硬盘；

## 文件管理★

1. 文件系统把磁盘的硬件特性和用户隔离开来，为用户提供“按名存取”的功能，是操作系统中负责存取和管理信息的模块，用统一的方式实现用户和系统信息的存储、检索、更新、共享和保护，为用户提供方便有效的文件使用和操作方法；

实现按名存取关键在于解决文件名和物理地址的转换；文件系统将磁盘抽象成容纳一组命名文件的设备；

1. 文件是由文件名标识的一组信息的集合，它的逻辑结构有流式文件和记录式文件；文件的物理结构是指文件在物理存储空间中存放方法和组织；文件的存储结构涉及块的划分、记录的排列、索引的组织、信息的搜索等许多问题，直接影响文件系统的性能；
2. 文件的物理结构有：顺序文件，链接文件，**索引文件**，直接文件（hash）；
3. 文件目录是实现按名存取的主要数据结构，文件系统的基本功能之一就是负责文件目录的建立、维护和检索，要求编排的目录便于查找、防止冲突，目录的检索方便迅速；实际的操作系统常使用树型目录结构；

目录可以按不同方法组织，有的把文件名、文件属性、磁盘地址放在一起组成FCB存放；有的仅存放文件名和inode号，其它信息则放到文件inode中；

1. 文件共享是指不同用户（进程）共同使用同一个文件，文件共享不仅为不同用户完成共同的任务所必须，而且还可以节省磁盘空间，减少文件复制；文件共享有静态共享（硬链接，按inode链接，只能链接文件，不能跨文件系统）、动态共享（进程间共享）、符号链接共享（软链接，按文件名链接，可跨文件系统）；
2. 磁盘空间管理：连续分配；非连续分配；空间管理方法：bitmap，空闲区表，空闲块链，成组空闲块；
3. 内存映射文件和虚拟内存系统两种特殊类型的文件；

**内存映射文件**把进程的虚地址空间与某一个磁盘文件关联起来，使得进程对文件的存取转化为对关联内存区的访问，消除两者操作的差异，简化编程；

**虚拟文件系统**是为了让一个操作系统同时支持多种文件系统，对多种具体文件系统的共性进行抽象，定义一致的接口，而形成的一个与具体文件系统实现无关的虚拟的通用文件系统；

## 操作系统的安全与保护

## 网络和分布式操作系统