系统文件打开表，

文件读写指针不存放在索引节点中

系统文件打开表主要存放文件位置，同时存放文件索引结点的指针

内核对所有打开的文件的文件维护有一个系统级打开文件表（open file table），并将表格中各条目称为打开文件句柄（open file handle）。一个打开文件句柄存储了与一个打开文件相关的全部信息，如下所示：

1. **当前文件偏移量（调用read()和write()时更新，或使用lseek()直接修改）**

2. 打开文件时所使用的状态标识（即，open()的flags参数）

3. 文件访问模式（如调用open()时所设置的只读模式、只写模式或读写模式）

4. 与信号驱动相关的设置

5. 对该文件**i-node对象的引用**

6. 文件类型（例如：常规文件、套接字或FIFO）和访问权限

7. 一个指针，指向该文件所持有的锁列表

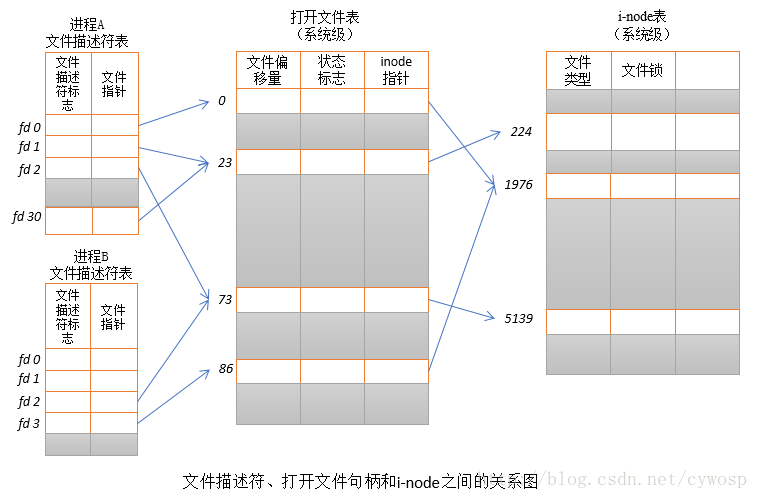
8. 文件的各种属性，包括文件大小以及与不同类型操作相关的时间戳

1. 由于进程级文件描述符表的存在，不同的进程中会出现相同的文件描述符，它们可能指向同一个文件，也可能指向不同的文件

2. **两个不同的文件描述符，若指向同一个打开文件句柄，将共享同一文件偏移量。**因此，如果通过其中一个文件描述符来修改文件偏移量（由调用read()、write()或lseek()所致），那么从另一个描述符中也会观察到变化，无论这两个文件描述符是否属于不同进程，还是同一个进程，情况都是如此。

3. 要获取和修改打开的文件标志（例如：O\_APPEND、O\_NONBLOCK和O\_ASYNC），可执行fcntl()的F\_GETFL和F\_SETFL操作，其对作用域的约束与上一条颇为类似。

4. 文件描述符标志（即，close-on-exec）为进程和文件描述符所私有。对这一标志的修改将不会影响同一进程或不同进程中的其他文件描述符



GBN信道传输速率计算

很容易联想到实际上要求的是信道利用率最高情况下发送窗口的大小（因为GBN协议的发送窗口W跟帧序号比特数n的关系为

。

下面找出信道利用率和发送窗口大小之间的联系。

在数据传输速率一定的条件下，**信道利用率可以等同于**

为了保证信道利用率最高为100%，那么我们必须保证发送方时刻有数据发出。

**对于发送方来说，从第一个发送帧发出到第一个发送帧的确认帧返回，把这个期间定义为发送周期**。

**如果把一个发送周期内发送方可以发出的帧的个数定为发送窗口的大小，那么此时信道利用率最高为100%。**

也就是说在一个发送周期中，发送方正好把一个发送窗口中的帧全部发出去了，没有闲置等待的时间。

设帧长为P，数据传输速率为B，单向传播时延为T。发送周期为 (P/B+T) \* 2信道利用率最高时，发送窗口大小为(P/B + T) \* 2 \* B / P = (1 + T \* B / P) \* 2由上式易知，

发送窗口大小是帧长P的单调递减函数。

取帧长 P =128 Byte**（这里取数据帧长的最小值）**，得到发送窗口大小的最大值，

也就得到了能够满足要求的帧序号比特数的最小值。

CF和OF标志位判断

CF是无符号数溢出标志，OF是有符号数溢出标志。

1、CF的判断

①加法

十进制角度，如果两无符号数相加，结果大于2^n-1（n为位数），则CF=1，否则CF=0；

二进制角度，如果两无符号数相加，最高位向前有进位，则CF=1，否则CF=0。

②减法

十进制角度，如果两无符号数相减，减数大于被减数（也即结果不在0—2^n-1内），则CF=1，否则CF=0；

二进制角度，如果两无符号数相减，最高位向前游借位，则CF=1，否则CF=0。

2、OF的判断

①加法

十进制角度，如果两有符号数相加，结果不在-2^(n-1)~2^(n-1)-1内，则OF=1，否则OF=0；

二进制角度，如果两有符号数同号，而相加结果与之异号，则OF=1，否则OF=0。

②减法

十进制角度，如果有符号数相减结果在-2^(n-1)~2^(n-1)-1内，则OF=1，否则OF=0；

二进制角度，如果两个数异号，而相减结果与被减数符号相反，则OF=1，否则OF=0。

CSMA/CA解决隐蔽站问题

RTS/CTS协议即请求发送/允许发送协议，相当于一种握手协议，主要用来解决"隐藏终端"问题。"隐藏终端"（Hidden Stations）是指，基站A向基站B发送信息，基站C未侦测到A也向B发送，故A和C同时将信号发送至B，引起信号冲突，最终导致发送至B的信号都丢失了。"隐藏终端"多发生在大型单元中（一般在室外环境），这将带来效率损失，并且需要错误恢复机制。当需要传送大容量文件时，尤其需要杜绝"隐藏终端"现象的发生。

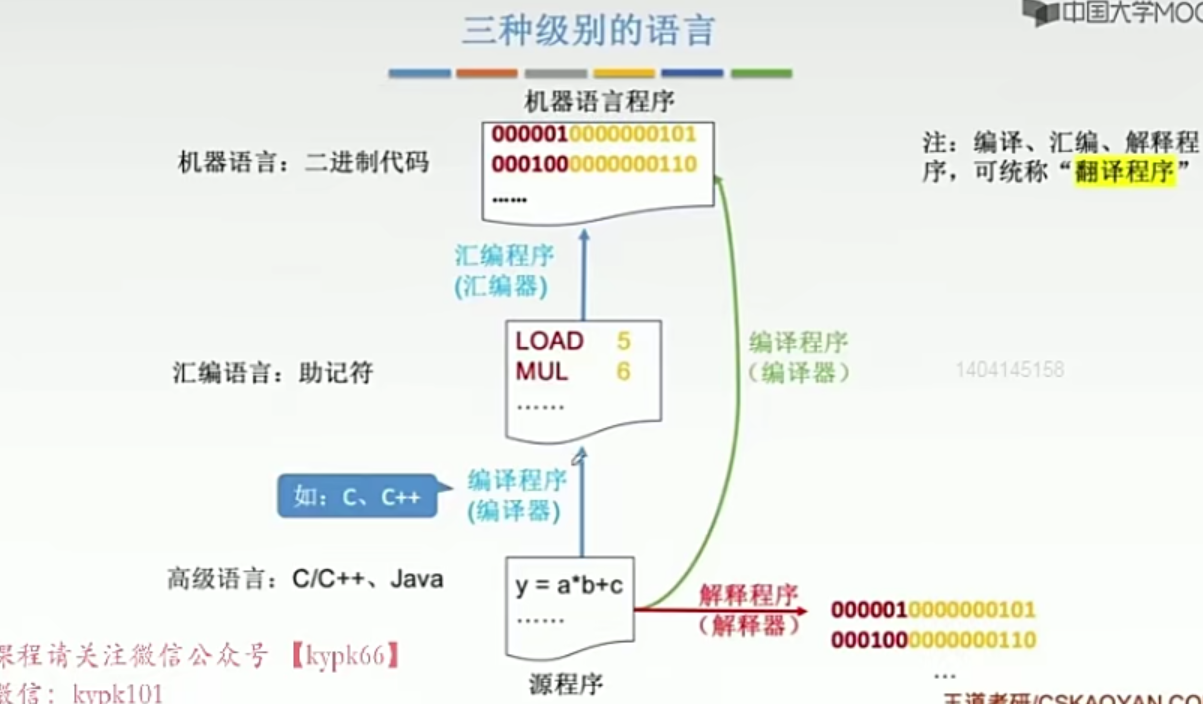
IEEE802.11提供了如下解决方案。

在参数配置中，若使用RTS/CTS协议，同时设置传送上限字节数----一旦待传送的数据大于此上限值时，即启动RTS/CTS握手协议：首先，A向B发送RTS信号，表明A要向B发送若干数据，B收到RTS后，向所有基站发出CTS信号，表明已准备就绪，A可以发送，其余基站暂时"按兵不动"，然后，A向B发送数据，最后，B接收完数据后，即向所有基站广播ACK确认帧，这样，所有基站又重新可以平等侦听、竞争信道了。

Cache中LRU控制位位数

对于n路相连的缓存来说，LRU每个缓存块需要的位数为log2(n)，那么每个set需要的位数就为n\*log2(n)。

编译、汇编和解释



编译程序和解释程序的区别

1. 编译型语言的源代码有错误编译不通过，无法生成可执行代码，更无法执行程序；解释型语言只有执行时才会判断是否出错，即使一句出错，也可以继续执行下一句。也即**编译程序将高级语言程序翻译成可执行文件，而解释程序一句一句翻译成可执行二进制代码**

2. **编译型语言执行效率上大大优于解释型**，主要因为编译器在编译过程中会根据不同平台自动优化目标代码，且特点为1次编译，N次运行，另外强类型的程序安全性高；解释型语言无上述过程，逐语句翻译造成执行效率低下，每次执行都会重复解释一遍，并且**安全性低**。

编译型程序适合对通用性，重复性，高效性有要求的系统，如开发操作系统；相比解释型语言更具灵活性，如开发网站前台页。

三态门和MUX

MUX（数据选择器/多路选择器）

数据选择器(data selector) 根据给定的输入地址代码，从一组输入信号中选出指定的一个送至输出端的组合逻辑电路

**工作原理是：**给A1A0一组信号 比如1 0 那么就相当于给了他一个2进制数字2 也就相当于选通了D2这个输入端，这个时候 输出Y 输出的就是D2的信号；D2是什么，Y就输出什么

三态门（用于控制内部组件数据何时流出）

计算机中的记忆元件由触发器组成，而触发器只有两个状态。即“0”态和“1”态，所以每条信号线上只能传送一个触发器的信息。如果要在一条信号线上连接多个触发器，而每个触发器可以根据需要与信号线连通或断开，当连通时可以传送“0”或“1”,断开时对信号线上的信息不产生影响，就需要一个特殊的电路加以控制，此电路即为三态输出电路，又称为三态门。

Buffer和Cache的区别

**1.buffer**

　　将数据写入到内存里，这个数据的内存空间在Linux系统里一般被称为缓冲区（buffer），例如：写入到内存buffer缓冲区，即写缓冲。

　　为了提高写操作性能，数据在写入最终介质或下一层级介质前会合并放在缓冲区中。这样会增加数据持久**写的延时**，因为第一次写入缓冲区后，在向下写入数据之前，还要等后续写入，以便**凑够数据或者定时写入到永久存储介质中**。

**2.cache**

　　从内存里读取数据，这个存数据的内存空间在Linux系统里一般称为缓存区（cache），例如：从内存cache读取，即**读缓存**。

　　操作系统用缓存提高（cache）文件系统的读性能和内存分配性能，应用程序使用缓存也是为了提升访问效率。将**经常访问的操作结果保存在缓存中可备随时使用**，而非总是 执行读磁盘取数据等开销较高的操作。

各种排序

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 时间 | 空间 | 稳定性 | 比较次数 | 移动次数 |
| 插入排序 | 直接插入排序 | 最坏n²  最好O(1) | O(1) | 稳定 | 最好情况O(n)  最坏∑i=O(n²) | 同比较次数 |
| 折半插入排序 | O(n²) | O(1) | 稳定 | 与初始状况无关  O(nlog2n) | 与直接插入排序相同 |
| 希尔排序 | 依赖于增量函数 | O(1) | 不稳定 | 与初始状况无关 | 与初始状况无关 |
| 交换排序 | 冒泡排序 | 最坏O(n²)  最好O(1) | O(1) | 稳定 | 最好情况O(n-1)  最坏O(n²) | 与初始状况有关 |
| 快速排序 | O(nlog2n)  最坏O(n²) | 最坏情况O(n)  平均O(log2n) | 不稳定 | 最好情况O(nlog2n)  最坏O(n²) | 与初始状况有关 |
| 选择排序 | 简单选择排序 | O(n²) | O(1) | 不稳定 | 与初始状况无关  始终是O(n²) | 最好移动0次  最坏移动O(n)次 |
| 堆排序 | O(log2n) | O(1) | 不稳定 | 插入和删除元素(**一次调整**):时间O(log2n)  **建堆**复杂度：O(n) | |
|  | 归并排序 | O(nlog2n) | O(n) | 稳定 | 与初始状况无关 | 与初始状况无关 |
|  | 基数排序 | O(d(n+r)) | O(r) | 稳定 | 与初始状况无关  一趟分配O(n)，一趟收集O(r)，一共需要d躺分配和收集 | |

不稳定的算法：

希简堆（洗剪吹）

与初始状态有关的算法：

直接插入、冒泡、快排（直接快跑）

设备分配数据结构

2 设备分配的数据结构：设备控制表（DCT）、控制器控制表（COCT）、通道控制表

（CHCT）、系统设备表（SDT）

**系统设备表（SDT）**：整个系统只有一张SDT，记录所有物理设备情况，每个物理设备

占一个表目

**设备控制表（DCT）**：**一个设备控制表表征一个设备**，各个表项就是设备的各个属性；同时表项中需要有一项指出该设备的控制器控制表（COCT）的指针

**控制器控制表（COCT）**：现代OS的IO都用通道控制，而设备控制器需要请求通道为其服务，所以有个表项指向相应通道控制表（CHCT）的指针

**通道控制表（CHCT）：**因为**一个通道可以为多个设备控制器服务**，所以CHCT必有一个指针，指向一个表，表上的信息表达的是CHCT提供服务的那几个设备控制器。

**CHCT与COCT是一对多的关系**

设备分配的过程中访问数据结构按**先后顺序为**：

SDT -> DCT -> COCT -> CHCT

**山东（SD）大（D）葱（CO）存货（CH）**