牛客网刷题笔记

算法

Seta(希腊字母，打不出来)表示法

上下限的时间复杂度都相同是

用它表示

设函数 f (n)代表某一算法在输入大小为 n 的情况下的工作量（效率），则在 n 趋向很大的时候，我们将 f (n)与另一行为已知的函数 g(n)进行比较：（假定和都是非负单调的，且极限存在）

a) 如果 lim f(n)/g(n)=0，则称 f(n)在数量级上严格小于 g(n)，（小 o 表示法）

b) 如果 lim f(n)/g(n)=正无穷，则称 f(n)在数量级上严格大于 g(n)

c) 如果 lim f(n)/g(n)=C，这里 C 为非 0 常数，则称 f(n)在数量级上等于 g(n)，即 f(n)和 g(n)是同一个数量级的函数，（Θ表示法）

d) 如果 f(n)在数量级上小于等于 g(n)，（大 O 表示法）

e) 如果 f(n)在数量级上大于等于 g(n)，（大Ω表示法）

C 语言

C-realloac

C 语言可以通过 realloc 对一个地址指向的内存再分配

```
    void *realloc(void *ptr, size_t size)
```

尝试重新调整之前调用 **malloc** 或 **calloc** 所分配的 **ptr** 所指向的内存块的大小。

- **ptr** -- 指针指向一个要重新分配内存的内存块，该内存块之前是通过调用 malloc、calloc 或 realloc 进行分配内存的。如果为空指针，则会分配一个新的内存块，且函数返回一个指向它的指针。
- **size** -- 内存块的新的大小，以字节为单位。如果大小为 0，且 ptr 指向一个已存在的内存块，则 ptr 所指向的内存块会被释放，并返回一个空指针。

内存分配

```
int main()
{
    char *p = "hello,world";
    return 0;
}
p 和"hello,world"存储在内存哪个区域？
RES: 栈，只读存储区
```

（1）从静态存储区域分配：

内存在程序编译时就已经分配好，这块内存在程序的整个运行期间都存在。速度快、不容易出错，因为有系统会善后。例如全局变量，static 变量等。

（2）在栈上分配：

在执行函数时，函数内局部变量的存储单元都在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

（3）从堆上分配：

即动态内存分配。程序在运行的时候用 malloc 或 new 申请任意大小的内存，程序员自己负责在何时用 free 或 delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活。如果在堆上分配了空间，就有责任回收它，否则运行的程序会出现内存泄漏，另外频繁地分配和释放不同大小的堆空间将会产生堆内碎块。

一个 C、C++程序编译时内存分为 5 大存储区：堆区、栈区、全局区、文字常量区、程序代码区。

计算机网络

默认端口

ftp: 对应两种模式，**21 端口用于连接，20 端口用于传输数据**

进行 FTP 文件传输中，**客户端首先连接到 FTP 服务器的 21 端口，进行用户的认证，认证成功后，要传输文件时，服务器会开一个端口为 20 来进行传输数据文件。**

也就是说，端口 20 才是真正传输所用到的端口，端口 21 只用于 FTP

的登陆认证。我们平常下载文件时，会遇到下载到 99%时，文件不完成，不能成功的下载。其实是因为文件下载完毕后，还要在 21 端口再行进行用户认证，而下载文件的时间如果过长，客户机与服务器的 21 端口的连接会被服务器认为是超时连接而中断掉，就是这个原因。解决方法就是设置 21 端口的响应时间。

网络体系结构

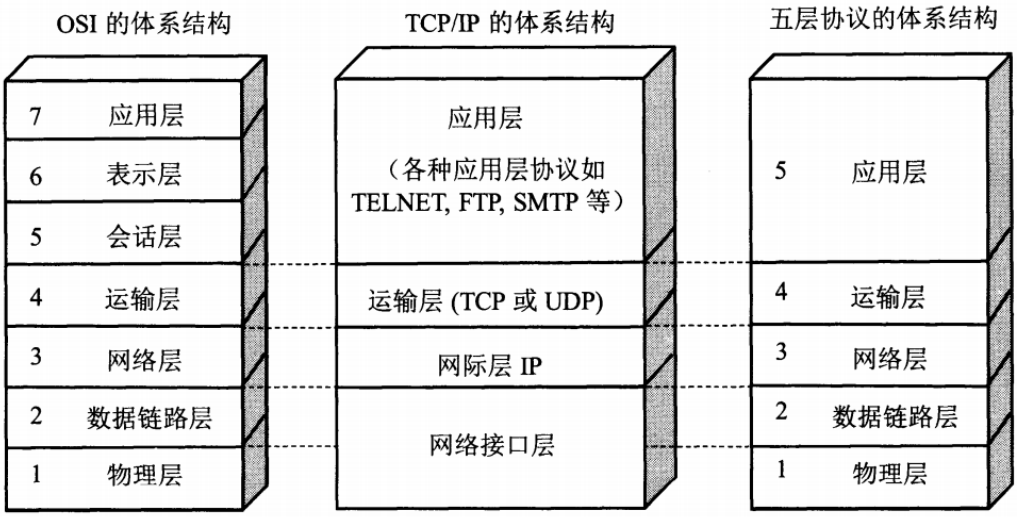


图 1-18　计算机网络体系结构

协议

STP（生成树协议）：是按照树的结构来构造网络拓扑，消除网络中的环路，避免由于环路的存在而造成广播风暴问题。

Tcp

tcp 按序传输 但不一定按序到达

udp

UDP 最大载荷为 1472

tcp vs. udp

1. TCP 粘包,就是发送方发送的多个数据包,到接收方后粘连在一起,导致数据包不能完整的体现发送的数据。
2. UDP 具有消息边界，不存在粘包问题。

TCP 协议中需要三次握手和四次挥手，UDP 不需要

OS

内存管理方案

- 连续分配:
  - 单一连续分配

    能用于单用户、单任务的 OS 中。

    将内存分为系统区（内存低端，分配给 OS 用）和用户区（内存高端，分配给用户用）。采用静态分配方式，即作业一旦进入内存，就要等待它运行结束后才能释放内存。
  - 固定分区分配

    将内存空间划分为若干个固定大小的分区，除 OS 占一区外，

其余的一个分区装入一道程序。分区的大小可以相等，也可以不等，但事先必须确定，在运行时不能改变。即分区大小及边界在运行时不能改变。

- 动态分区分配

  又称为可变式分区分配，是一种动态划分存储器的分区方法。不事先将内存划分成一块块的分区，而是在作业进入内存时，根据作业的大小动态地建立分区，并使分区的大小正好适应作业的需要。因此系统中分区的大小是可变的，分区的数目也是可变的。

  包含 2 种类型：11、**基于顺序搜索的动态分区分配算法。2、1 基于索引搜索的动态分区分配算法。**

- 动态重定位分区分配

  与动态分区分配算法基本上相同，差别仅在于：在这种分配算法中，增加了紧凑的功能。通常，当该算法不能找到一个足够大的空闲分区以满足用户需求时，如果所有的小的空闲分区的容量总和大于用户的要求，这时便须对内存进行“紧凑”，将经“紧凑”后所得到的大空闲分区分配给用户。如果所有的小的空闲分区的容量总和仍小于用户的要求，则返回分配失败信息。

- 分段：
- 分页：
- 段页式：

多道程序·封闭性

指程序一旦开始执行，其计算结果就取决于自身，不受其他外部环境影响

多道程序·可再现性

程序多次执行的结果相同

操作系统的五大功能

1、处理器管理

处理器管理最基本的功能是处理中断事件，配置了操作系统后，就可对各种事件进行处理。处理器管理还有一个功能就是处理器调度，针对不同情况采取不同的调度策略。

2、存储器管理

存储器管理主要是指针对内存储器的管理。主要任务是分配内存空间，保证各作业占用的存储空间不发生矛盾，并使各作业在自己所属存储区中不互相干扰。

3、设备管理

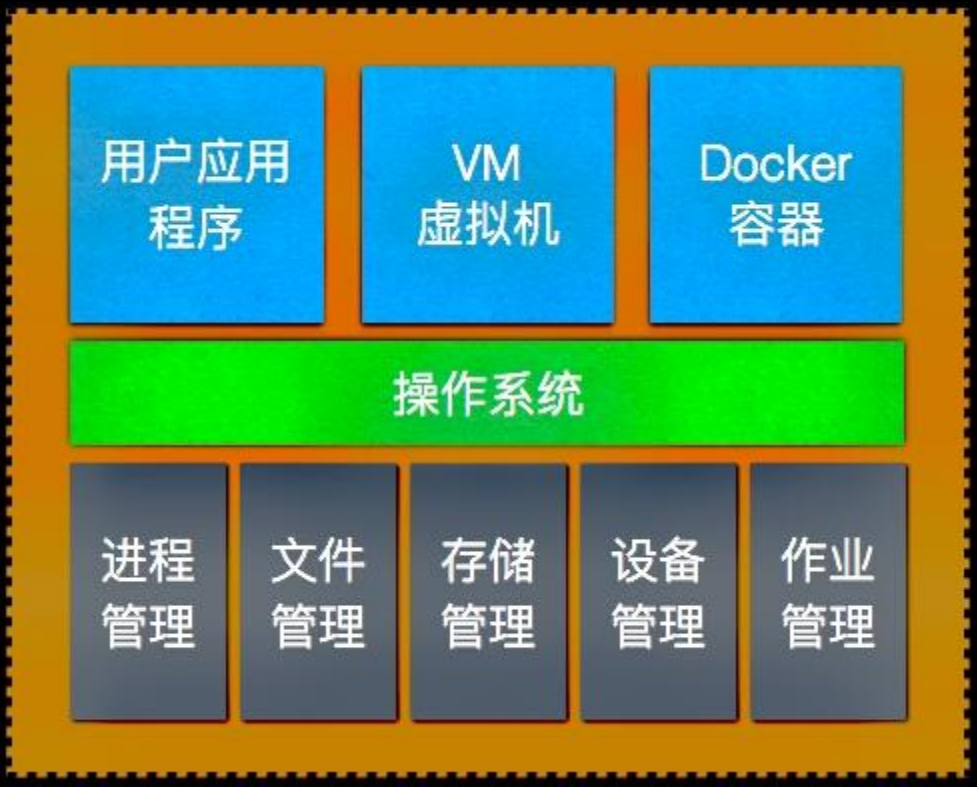
设备管理是指负责管理各类外围设备，包括分配、启动和故障处理等。主要任务是当用户使用外部设备时，必须提出要求，待操作系统进行统一分配后方可使用。

4、文件管理

文件管理是指操作系统对信息资源的管理。在操作系统中，将负责存取的管理信息的部分称为文件系统。文件管理支持文件的存储、检索和修改等操作以及文件的保护功能。

5、作业管理

每个用户请求计算机系统完成的一个独立的操作称为作业。作业管理包括作业的输入和输出，作业的调度与控制，这是根据用户的需要来控制作业运行的。



进程通信的方式

每个进程各自有不同的用户地址空间，任何一个进程的全局变量在另一个进程中都看不到，所以进程之间要交换数据必须通过内核，在内核中开辟一块缓冲区，进程 A 把数据从用户空间拷到内核缓冲区，

进程 B 再从内核缓冲区把数据读走，内核提供的这种机制称为进程间通信

不同进程间的通信本质：进程之间可以看到一份公共资源；而提供这份资源的形式或者提供者不同，造成了通信方式不同

- 管道

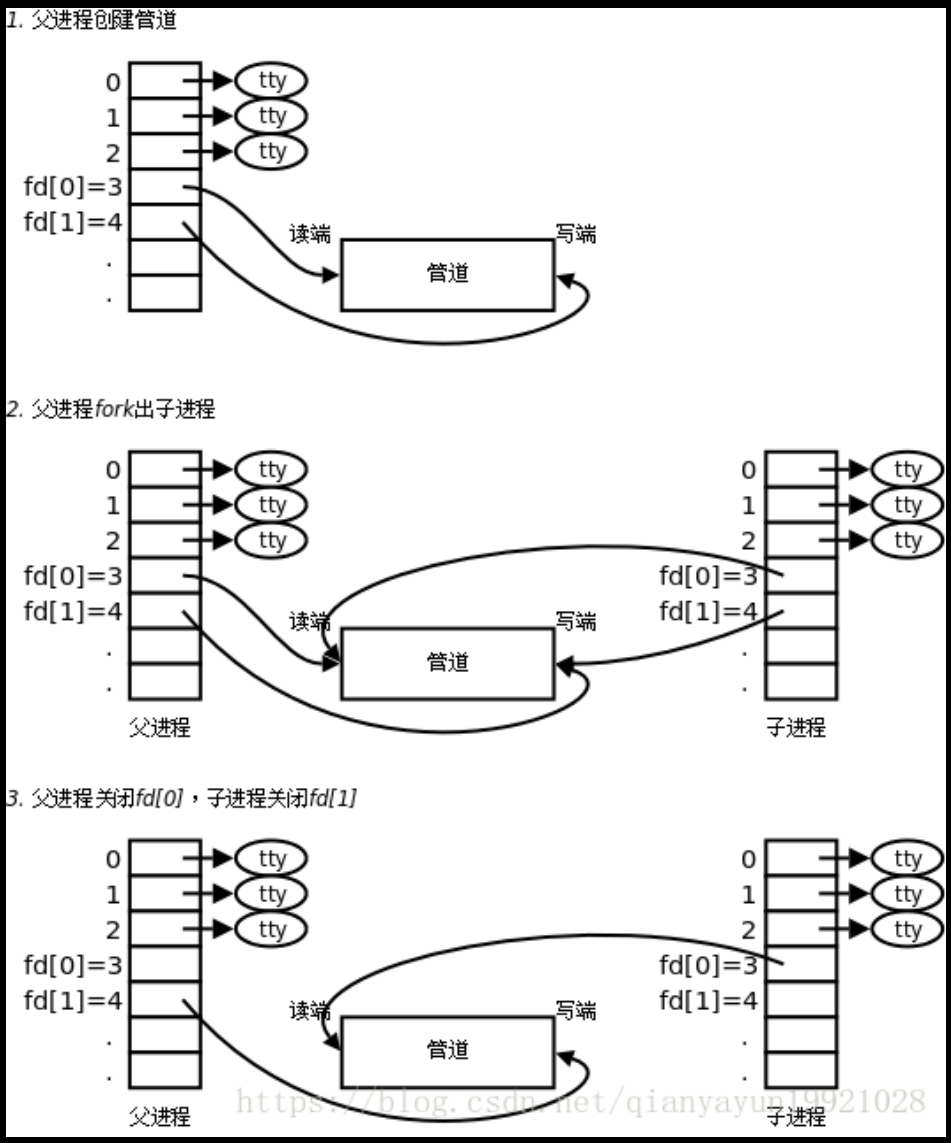
  管道是一种最基本的 IPC 机制，由 pipe 函数创建：

  ```
  int pipe(int filedes[2]);
  ```

  管道作用于**有血缘关系的进程之间**，通过 fork 来传递

  调用 pipe 函数时在内核中开辟一块缓冲区（称为管道）用于通信，它有**一个读端一个写端**，然后通过 filedes 参数传出给用户程序两个文件描述符，**filedes[0]指向管道的读端，filedes[1]指向管道的写端**（很好记，就像 0 是标准输入 1 是标准输出一样）。所以管道在用户程序看起来就像一个打开的文件，通过 read(filedes[0]);或者 write(filedes[1]);

  向这个文件读写数据其实是在读写内核缓冲区。pipe 函数调用成功返回 0，调用失败返回-1。



管道是用环形队列实现的，数据从写端流入从读端流出，这样就实现了进程间通信。

管道有一些限制：

1. 两个进程通过一个管道只能实现**单向通信(环形队列)——半双工：一个发给另一个，不能同时发。**
2. 管道的读写端通过打开的文件描述符来传递，因此要通信的两个进程必须从它们的公共祖先那里继承管道文件描述符。

3. 读取四种情况：

a) 读端不读，写端一直写

b) 写端不写，但是读端一直读

c) 读端一直读，且 fd[0]保持打开，而写端写了一部分数据不写了，并且关闭 fd[1]。，没有了读到 0，不会导致阻塞

d) 读端读了一部分数据，不读了且关闭 fd[0]，写端一直在写且 f[1]还保持打开状态。阻塞

4. 面向字节流

5. 管道随进程，进程在管道在，进程消失管道对应的端口也关闭，两个进程都消失管道也消失

- FIFO

Named pipe 有名管道

命名管道，它是一种文件类型。

1. FIFO 可以在**无关的进程之间交换数据**，与无名管道不同。
2. FIFO 有**路径名与之相关联**，它以一种**特殊设备文件形式**存在于文件系统中。

```
#include <sys/stat.h>
// 返回值：成功返回0，出错返回-1
int mkfifo(const char *pathname, mode_t mode);
```

mode 参数与 open 函数中的 mode 相同。一旦创建了一个 FIFO，就可以用一般的文件 I/O 函数操作它。

当 open 一个 FIFO 时，是否设置非阻塞标志（O_NONBLOCK）

的区别：

- 若没有指定 O_NONBLOCK（默认），只读 open 要阻塞到某个其他进程为写而打开此 FIFO。类似的，只写 open 要阻塞到某个其他进程为读而打开它。
- 若指定了 O_NONBLOCK，则只读 open 立即返回。而只写 open 将出错返回 -1 如果没有进程已经为读而打开该 FIFO，其 errno 置 ENXIO。

FIFO 的通信方式**类似于在进程中使用文件来传输数据，只不过 FIFO 类型文件同时具有管道的特性**。在数据读出时，FIFO 管道中同时清除数据，并且“先进先出”。

- 消息队列

  消息的链接表，存放在内核中。一个消息队列由一个标识符（即队列 ID）来标识。

  特点

  1. 消息队列是**面向记录**的，其中的消息具有**特定的格式以及特定的优先级**。
  2. 消息队列**独立于发送与接收进程**。进程终止时，消息队列及其内容并不会被删除。
  3. 消息队列可以实现**消息的随机查询**，消息不一定要以先进先出的次序读取，也可以按消息的类型读取。

```c
#include <sys/msg.h>
// 创建或打开消息队列：成功返回队列ID，失败返回-1
int msgget(key_t key, int flag);
// 添加消息：成功返回0，失败返回-1
int msgsnd(int msqid, const void *ptr, size_t size, int flag);
// 读取消息：成功返回消息数据的长度，失败返回-1
int msgrcv(int msqid, void *ptr, size_t size, long type,int flag)
// 控制消息队列：成功返回0，失败返回-1
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- 信号量

  信号量（semaphore）与已经介绍过的 IPC 结构不同，它是一个计数器。信号量用于实现进程间的互斥与同步，而不是用于存储进程间通信数据。

  特点

1. 信号量用于进程间同步，若要在进程间传递数据需要结合共享内存。
2. 信号量基于操作系统的 PV 操作，程序对信号量的操作都是原子操作。
3. 每次对信号量的 PV 操作不仅限于对信号量值加 1 或减 1，而且可以加减任意正整数。
4. 支持信号量组。

- 共享内存

  共享内存（Shared Memory），指两个或多个进程共享一个给定的存储区。

  特点

1. 共享内存是**最快的一种 IPC**，因为进程是直接对内存进行存取。
2. 因为多个进程可以**同时操作，所以需要进行同步**。
3. **信号量+共享内存通常结合在一起使用**，信号量用来同步对共享内存的访问。

- **socket 通信**(包括同一主机上的 socket 通信和不在同一主机上的远程 socket 通信)

  套接口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

6 种通讯方式总结

1. **管道：速度慢，容量有限**，只有**血缘进程**能通讯
2. FIFO：任何进程间都能通讯，但速度慢
3. 消息队列：**容量受到系统限制**，且要注意第一次读的时候，要考虑上一次没有读完数据的问题
4. **信号量：不能传递复杂消息，只能用来同步**
5. **共享内存区**：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了同一进程内的一块内存

大概是五类：**管道通信、消息队列、共享内存、信号量、socket**

ICP

IPC: Inter process Communication, **通过内核提供的缓冲区进行数据交换的机制。**

IPC 通信的方式:

1. pipe: 管道（最简单）
2. fifo: 有名管道
3. mmap: 打开一块共享的内存（速度最快）
4. 本地套接字: 最稳定
5. 信号: 携带信息量最小
6. 共享内存
7. 消息队列

PV

PV 操作: 一种实现进程互斥与同步的有效方法, 包含 P 操作与 V 操作。

P 操作: 使 S=S-1 ，若 S>=0 ，则该进程继续执行, 否则排入等待队列。

V 操作: 使 S=S+1 ，若 S>0 ,唤醒等待队列中的一个进程。

临界资源:同一时刻只允许一个进程访问的资源,与上面所说的 S 有

关联。

可变分区分派方案

最先适应算法：依次判定后找到第一个满足要求的哈

最佳适应算法：对空闲区按从小到大排序，第一个满足的就是啦

最差适应算法：对空闲区按从大到小排序，第一个满足的就是啦

固定式分区算法：是分区的

只需要进行一次比较就可以判定系统是否能满足作业对主存空间要求的算法：最差适应算法

死锁 4 条件

死锁是指多个进程因竞争资源而造成的一种僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

4 个条件：

1. **互斥条件：一个资源每次只能被一个进程使用**，即在一段时间内某 资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。
2. **请求与保持条件**：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源 已被其他进程占有，此时请求进程被阻塞，

但对自己已获得的资源保持不放。

3．不可剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能 由获得该资源的进程自己来释放（只能是主动释放）。

4．循环等待条件：若干进程间形成首尾相接循环等待资源的关系

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁

死锁解决

1．死锁避免

基本思想：系统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，如果分配后系统可能发生死锁，则不予分配，否则予以分配，这是一种保证系统不进入死锁状态的动态策略。

2．死锁预防

资源剥夺法允许一个进程强行剥夺其他进程所占有的系统资源。而撤销进程是强行释放一个进程已占有的系统资源，与资源剥夺法同理，都是通过破坏死锁的“请求和保持”条件来解除死锁。拒绝分配新资源只能维持死锁的现状，无法解除死锁。

设备独立性

设备独立性，即应用程序独立于具体使用的物理设备。为了实现设备独立性而引入了逻辑设备和物理设备这两个概念。

用户编程时使用的设备与实际使用的设备无关

引起创建进程的事件

1. 用户登录
2. 作业调度
3. 提供服务
4. 应用请求

Linux

Linux 重要目录

dev 设备信息

home 家目录

bin/sbin 可执行文件

etc 系统文件

lib/lib64 动态库/静态库

lost+found 丢失文件

opt 第三方文件（相对于操作系统）

root 管理员的家目录

usr 第三方头文件和库

var 系统日志文件/缓存文件

Linux 基础命令

ps（Process Status）：查看系统进程的情况

nestat：查看网络情况

df（disk free）：查看磁盘情况

ifconfig 命令用来查看和配置网络设备，当网络环境发生改变时可通过此命令对网络进行相应的配置

```
head -n k          # 打印前 k 行
head -n -k         # 打印除最后 k 行外的所有内容
tail -n k          # 打印最后 k 行
tail -n +k         # 从 k 行开始打印
```

rpm

不同 Linux 发行版用的是不同软件包系统，分为 Debian 的.deb 技术（Debian,Ubuntu）和 Red Hat 的 .rpm 技术

(Fedora,CentOS,redHat).

其中 rpm 是低级的 RedHat 软件包管理工具:

-i　　package_name　　安装软件包

-U　　packagefile　　更新软件包

-qa　　列出已安装的软件包列表

-q　　package-name　　判断软件包是否安装

-qf　　file_name　　判断某个特定文件是由哪个软件包负责安装的

Linux 下两个进程可以同时打开同一个文件

两个进程中分别产生生成两个独立的 fd

两个进程可以任意对文件进行读写操作，操作系统并不保证写的原子性

进程可以通过系统调用对文件加锁，从而实现对文件内容的保护

两个进程可以分别读取文件的不同部分而不会相互影响

一个进程对文件长度和内容的修改另外一个进程可以立即感知

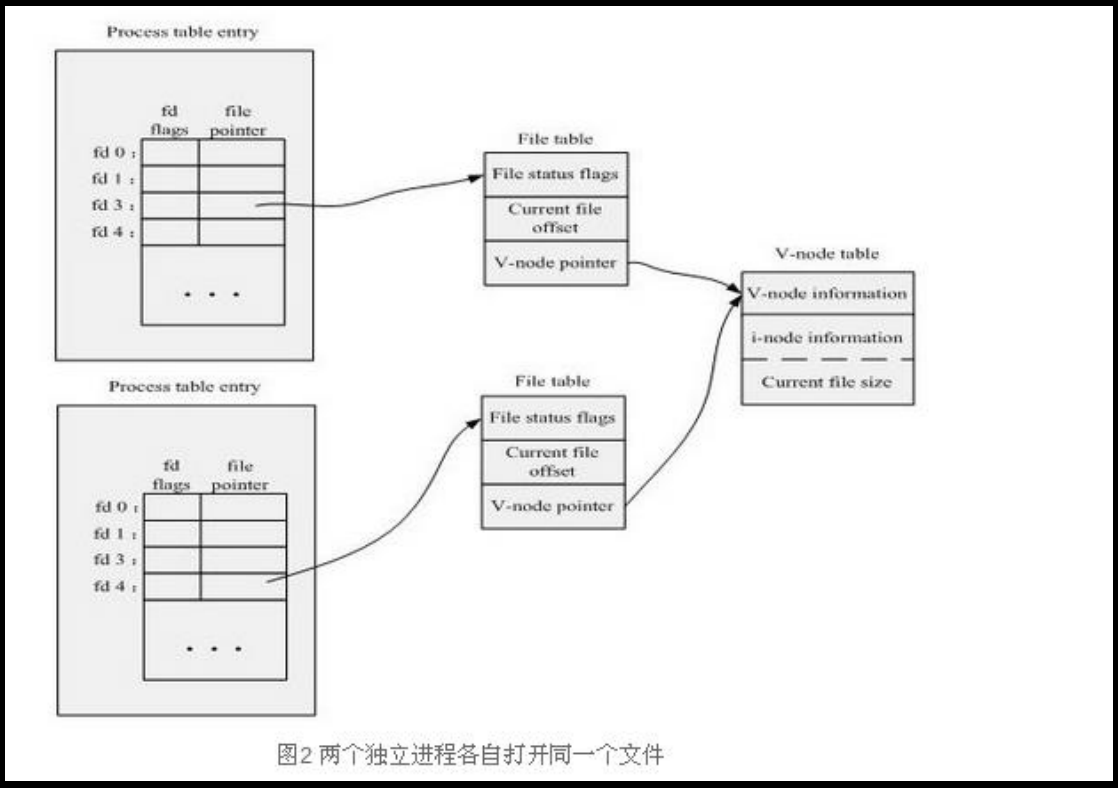


图2 两个独立进程各自打开同一个文件

Linux 文件权限

linux 文件的权限如 -rwxr-xr-x

第一位： - 代表是文件，第一位是 d 代表是目录

第 2～10 个字符当中的**每 3 个为一组**，左边三个字符表示**所有者权限**，中间 3 个字符表示与**所有者同一组的用户的权限**，右边 3 个字符是**其他用户的权限**。这三个一组共 9 个字符，代表的意义如下：

1. r(Read，读取)：对文件而言，具有读取文件内容的权限；对目录来说，具有浏览目录的权

2. w(Write,写入)：对文件而言，具有新增、修改文件内容的权限；对目录来说，具有删除、移动目录内文件的权限。
3. x(eXecute，执行)：对文件而言，具有执行文件的权限；对目录了来说该用户具有进入目录的权限。
4. —：表示不具有该项权限。

chown 将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户 ID；组可以是组名或者组 ID；文件是以空格分开的要改变权限的文件列表，支持通配符。系统管理员经常使用 chown 命令，在将文件拷贝到另一个用户的名录下之后，让用户拥有使用该文件的权限。
命令格式：chown [选项]... [所有者][:[组]] 文件...

Ll 命令：-rwxrw-r--1 root root 1213 Feb 2 09:39 abc

1. 10 个字符确定不同用户能对文件干什么
2. 第一个字符代表**文件（-）、目录（d），链接（l）**
3. 其余字符每 3 个一组（rwx），读（r）、写（w）、执行（x）
4. 第一组 rwx：文件所有者的权限是读、写和执行
5. 第二组 rw-：与文件所有者同一组的用户的权限是读、写但不能执行
6. 第三组 r--：不与文件所有者同组的其他用户的权限是读不能

写和执行

7. 也可用数字表示为：r=4，w=2，x=1 因此 rwx=4+2+1=7

8. 1 表示连接的文件数

9. root 表示用户

10. root 表示用户所在的组

11. 1213 表示文件大小（字节）

12. Feb 2 09:39 表示最后修改日期

13. abc 表示文件名

chmod

语法格式：

1. chmod [模式] 文件

2. chmod [八进制] 文件

- 操作对象：u 主用户 g 同组用户 其他用户 a(ugo) 所有用
- 权限类别： **r (4) 读 w(2) 写 x(1)执行 所有权限（7）**
- 权限设定：**+增加权限 - 取消权限 = 唯一设定权限**

Vi editor

yy 实现复制一行数据

pp 实现粘贴作用

dd 删除一整行

grep

是一种强大的文本搜索工具，全称是 Global Regular Expression Print

^表示开头，$表示结束

-E 表示使用扩展表达式

 -e<范本样式>或--regexp=<范本样式> 指定字符串做为查找文件内容的范本样式。

  -E 或--extended-regexp 将范本样式为延伸的普通表示法来使用。

Shell 脚本

shell 中命令行 参数:**$0 表示脚本名,$1 -$9 表示后面的参数**

终止前后台进程

终止一个前台进程用 ctrl+C;

终止一个后台进程:

1、使用 kill 命令;

2、使用 fg 命令将后台进程变为前台进程，然后 ctrl+C。

Linux 文件

文件操作的唯一依据是：**文件句柄**

Linux 打印

1. **lprm** 命令用于将一个工作由打印机贮列中移除
2. **lpq** 命令用于查看一个打印队列的状态，该程序可以查看打印机队列状态及其所包含的打印任务。
3. **lpd** 命令 是一个常驻的打印机管理程序，它会根据 /etc/printcap 的内容来管理本地或远端的打印机。
4. **lpr**(line printer，按行打印)实用程序用来将一个或多个文件放入打印队列等待打印。

关机重启

关机命令有 **halt** 、**init 0**、 **poweroff** 、 **shutdown -h** 时间，其中 shutdown 是最安全的

重启命令有 **reboot**, **init 6**, **shutdow -r** 时间

1、shutdown 执行它的工作是送信号〔signal〕给 init 程序，要求它改变 runlevel。Runlevel 0 被用来停机〔halt〕，runlevel 6

是用来重新激活〔reboot〕系统，而 runlevel 1 则是被用来让系统进入管理工作可以进行的状态；这是预设的。

2、其实 halt 就是调用 shutdown -h。halt 执行时，杀死应用进程，执行 sync 系统调用，文件系统写操作完成后就会停止内核。

3、reboot 的工作过程差不多跟 halt 一样，不过它是引发主机重启，而 halt 是关机。它的参数与 halt 相差不多。

Spinlock 自旋锁

spinlock 又称自旋锁,线程通过 busy-wait-loop 的方式来获取锁,**任时刻只有一个线程能够获得锁，其他线程忙等待直到获得锁**。

spinlock 在多处理器多线程环境的场景中有很广泛的使用，一般要求使用 spinlock 的**临界区尽量简短**，这样获取的锁可以尽快释放，以满足其他忙等的线程。Spinlock 和 mutex 不同，spinlock **不会导致线程的状态切换(用户态->内核态)**，但是 spinlock 使用不当(如临界区执行时间过长)会导致 cpu busy 飙高。

Java

赋值

只有当给变量赋值的时候才会分配内存空间——变量创建但是没有

赋值是不会分配内存空间的。

并发编程

DK 提供的用于并发编程的同步器：

1. Semaphore
2. CyclicBarrier
3. CountDownLatch

HashMap

HashMap 中是用哪些方法来解决哈希冲突的

链地址法

synchronized

synchronized 保证三大性，**原子性**，**有序性**，**可见性**，

volatile

保证**有序性**，**可见性**，不能保证原子性

volatile 到底做了什么：

- **禁止了指令重排**
- **保证了不同线程对这个变量进行操作时的可见性，即一个线程**

修改了某个变量值，这个新值对其他线程是立即可见的

- 不保证原子性（线程不安全）

synchronized 关键字和 volatile 关键字比较：

- volatile 关键字是线程同步的轻量级实现，所以 volatile 性能肯定比 synchronized 关键字要好。但是 volatile 关键字只能用于变量而 synchronized 关键字可以修饰方法以及代码块。synchronized 关键字在 JavaSE1.6 之后进行了主要包括为了减少获得锁和释放锁带来的性能消耗而引入的偏向锁和轻量级锁以及其它各种优化之后执行效率有了显著提升，实际开发中使用 synchronized 关键字的场景还是更多一些。
- 多线程访问 volatile 关键字不会发生阻塞，而 synchronized 关键字可能会发生阻塞
- volatile 关键字能保证数据的可见性，但不能保证数据的原子性。synchronized 关键字两者都能保证。
- volatile 关键字主要用于解决变量在多个线程之间的可见性，而 synchronized 关键字解决的是多个线程之间访问资源的同步性。

深入 volatile 关键字的介绍

1）被 volatile 关键字修饰的实例变量或者类变量具备两层语义：

- 保证了不同线程之间对共享变量的可见性，

- 禁止对 volatile 变量进行重排序。

2）volatile 和 synchronized 区别

- 使用上区别:

1. volatile 关键字只能用来修饰实例变量或者类变量，不能修饰方法已及方法参数和局部变量和常量。
2. synchronized 关键字不能用来修饰变量，只能用于修饰方法和语句块。
3. volatile 修饰的变量可以为空,同步块的 monitor 不能为空。

- 对原子性的保证

1. volatile 无法保证原子性
2. synchronizde 能够保证。因为无法被中途打断。

- 对可见性的保证

1. 都可以实现共享资源的可见性，但是实现的机制不同，synchronized 借助于 JVM 指令 monitor enter 和 monitor exit ，通过排他的机制使线程串行通过同步块，在 monitor 退出后所共享的内存会被刷新到主内存中。volatile 使用机器指令(硬编码)的方式,“lock”迫使其他线程工作内存中的数据失效，不得不主内存继续加载。

- 对有序性的保证

1. volatile 关键字禁止 JVM 编译器已及处理器对其进行重排序，能够保证有序性。
2. synchronized 保证顺序性是串行化的结果，但同步块里的语

句是会发生指令从排。

- 其他：

1. volatile 不会使线程陷入阻塞
2. synchronized 会会使线程进入阻塞。

Switch 支持的数据类型

jdk1.7 之前 byte,short ,int ,char(没有 long)

jdk1.7 之后加入 String

基本类型

+= 会自动强转（自动装箱功能），但是 + 必须要手动强转

b=(byte)(a+b)

Try

finally 块中的 return 语句会覆盖 try 块\catch 中的 return 返回

```java
public static int f(){
    try {
        String s="";
        s = null;
        s.length();
        return 1;
    }catch (Exception e){
        e.printStackTrace();
        return 2;
    }finally {
        return 3;
    }
}
```

覆盖之前的

跳出多重循环

定义标签 L，然后 break L

```java
a:
for (int i = 0; i < 100; i++) {
    System.out.println("i: "+i);
    for (int j =0;j<100;j++){
        System.out.println(i);
        break a;
    }
}
```

```java
for (int i = 0; i < 100; i++) {
    System.out.println("i: "+i);
    b:
    for (int j =0;j<100;j++){
        System.out.println(i);
        break b;
    }
}
```

递归的优缺点

优点：

1. 简洁

2.在树的前序，中序，后序遍历算法中，递归的实现明显要比循环简单得多。

缺点：

1.递归由于是函数调用自身，而函数调用是有**时间和空间**的消耗的：每一次函数调用，都需要在内存栈中分配空间以保存参数、返回地址以及临时变量，而往栈中压入数据和弹出数据都需要时间。->**效率**

2.递归中很多计算都是**重复**的，由于其本质是把一个问题分解成两个或者多个小问题，多个小问题存在相互重叠的部分，则存在重复计算，如 fibonacci 斐波那契数列的递归实现。->效率

3.**调用栈可能会溢出**，其实每一次函数调用会在内存栈中分配空间，而每个进程的栈的容量是有限的，当调用的层次太多时，就会超出栈的容量，从而导致栈溢出。->性能