**char str[]与char \*str的区别**

char str[]="hello";

第一个表达式表示的是在动态变量区中开辟一个能连续放6个字符的数组，数组名称是str. 而赋值运算符右边是一个字符串常量，这个字符串常量是存放在常量区的，这个表达式的意思就是将“hello”这个字符串常量**拷贝**到刚才开辟的数组中（重点）。

C语言规定，表达式如果是一个数组名，则代表的意思是该数组的起始地址，如果这个数组在一个函数中定义，如果以数组名返回时（以变量返回时可以，因为返回的是值），因为数组在函数中定义，是个局部变量，函数返回之后，这个数组所占用的空间就被释放掉了，数组也被破坏掉了，因此返回的数组名也就没有意义，不能被其主调函数使用了。

如果我们这样写：

char \*foo()

{

char str[]="hello";

return str;

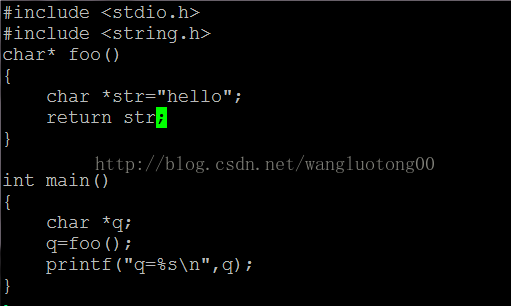
}

在编译的时候就会出现警告：函数返回局部变量的地址

再来看下char \*str="hello"

这个表达式的意思是在动态变量区中开辟一个存放指针的存储单元，指针变量名是str, 指针变量地址直接指向存储在常量区（与全局变量在一个区，程序运行过程中一直在，函数返回时不会被释放）的"hello"字符串常量，意思是指针变量的值就是HELLO字符串常量的地址，。把字符串“hello”的地址值拷贝到刚才的存储单元中，即指针变量str的初值是字符串“hello”的地址。这时如果char \*str="hello"定义在一个函数中并且以return str返回，因为str是一个变量名，返回的仅仅是指针变量的值，所以在其他函数中可以使用该值，照样能够访问到“hello”这个字符串，例如：

这里的str是一个变量，里面存放的是指向h的地址，指针变量str本身也是一个变量，也需要占据存储空间（32位系统下占4个字节），同样也有地址

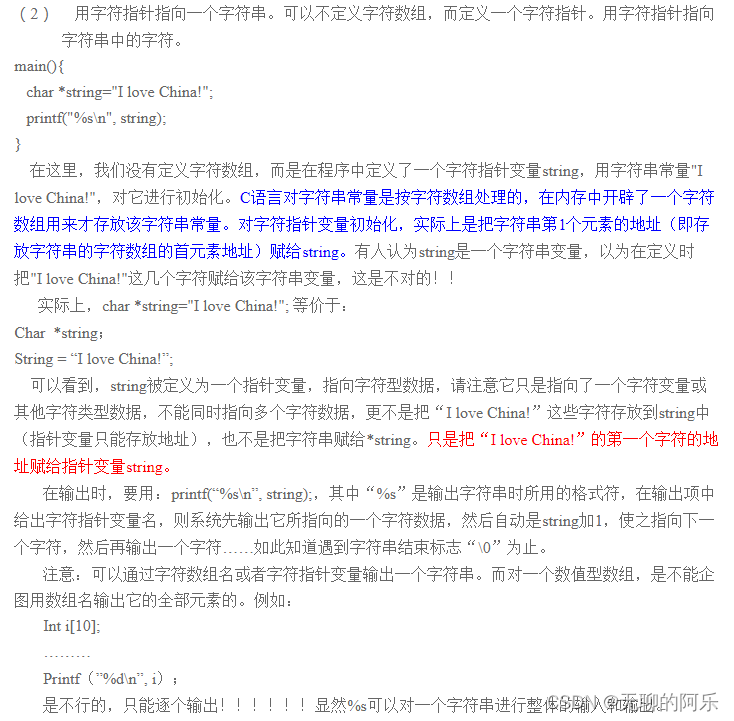


打印：hello

Print(“%d”,\*q) 打印：104 h的ascii字符位104

Print(“%c”,\*q) 打印：h 指针变量指向的第一个值

%s在输出项中给出字符指针变量名，系统会先输出变量名所指向的一个是字符数据（该模式只有字符串管用），然后自动string+1,指向下一个字符，指导遇到“/0”为止。



**（1）char ss[ ]定义了一个数组，ss可认为是一个常指针，ss不可改变，但ss指向的内容可以发生改变。**

**（2）char \*p定义了一个可变指针，p可以指向其它对象。但对于char \*p=”abc“这样的情况，p指向的是常量，**存储在常量区,**故内容不能改变。**

char ss[]="C++";

ss[0]='c';                  *// 合法*

char \*p="C++";

p[0]='c';                   *// 合法但不正确*

# Socket

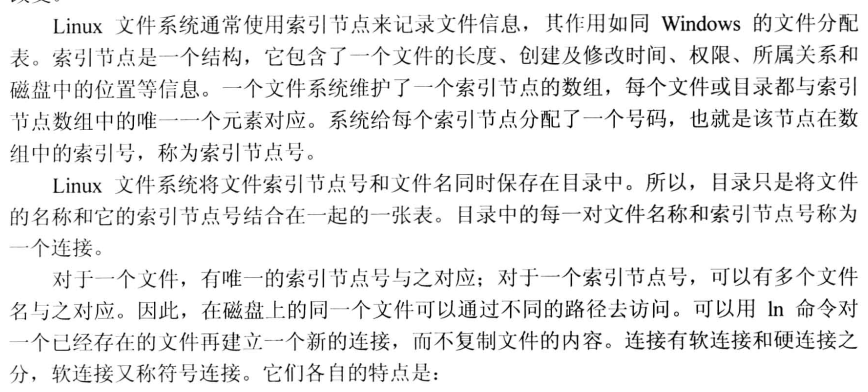
Socket是[应用层](https://so.csdn.net/so/search?q=%E5%BA%94%E7%94%A8%E5%B1%82&spm=1001.2101.3001.7020)与TCP/IP协议族通信的中间软件抽象层，它是一组接口。是TCP/IP网络的[API](https://so.csdn.net/so/search?q=API&spm=1001.2101.3001.7020)，Socket接口定义了许多函数或例程，程序员可以用它们来开发TCP/IP网络上的应用程序

# 值传递，地址传递

值传递:函数传值时,形参会首先根据其定义类型开辟新的空间,并将其参数值复刻到新的内存空间中,新的内存空间开辟也就意味着其存储地址的改变,而函数中对其参数的操作,也只是对于新空间内操作,并不会对函数外,传的值作影响,即两个是完全不同的空间.

地址传递:通常通过指针变量作为参数,由于指针变量是保存地址的变量,所以在通过指针变量去接收时,是传的地址,也就是说,在内存空间中,指向的是同一内存块,此时的操作会同时受到影响

**Linux文件系统**

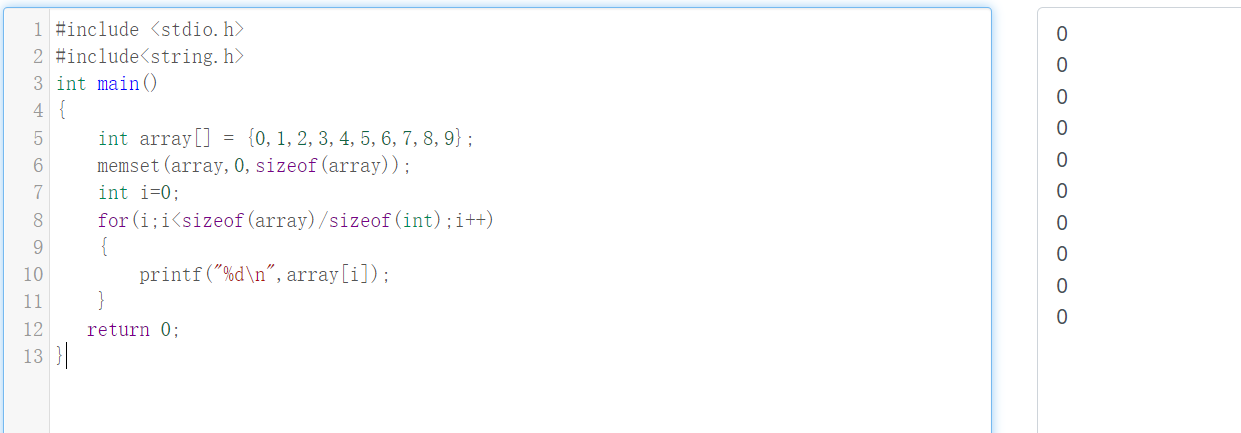
**寄存器**

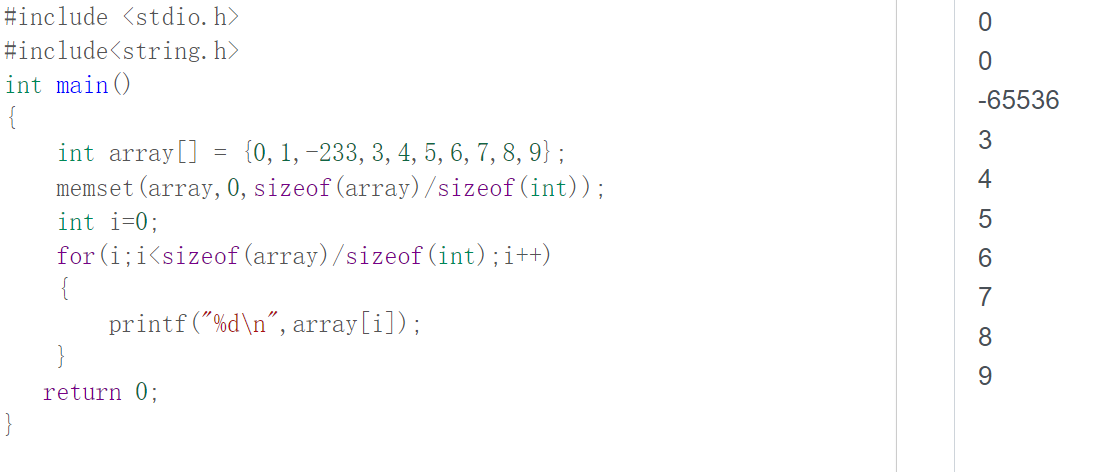
CPU 本身只负责运算，不负责储存数据。

数据一般都储存在内存之中，CPU 要用的时候就去内存读写数据。但是，CPU 的运算速度远高于内存的读写速度，为了避免被拖慢，CPU 都自带一级缓存和二级缓存。基本上，CPU 缓存可以看作是读写速度较快的内存。但是，CPU 缓存还是不够快，另外数据在缓存里面的地址是不固定的，CPU 每次读写都要寻址也会拖慢速度。

因此，除了缓存之外，CPU 还自带了寄存器（register），用来储存最常用的数据。也就是说，那些最频繁读写的数据（比如循环变量），都会放在寄存器里面，CPU 优先读写寄存器，再由寄存器跟内存交换数据。

**Memset**

传入地址 并对该地址 及其连续地址对应的值 赋对应字节长度的值 



-233前两个字节赋值为0

**DMA**

DMA(Direct Memory Access：直接内存存取)允许不同速度的硬件装置来沟通，而不需要依于 CPU 的大量中断负载，是一种可以大大减轻 CPU 工作量的数据转移方式。 需要DMA控制器

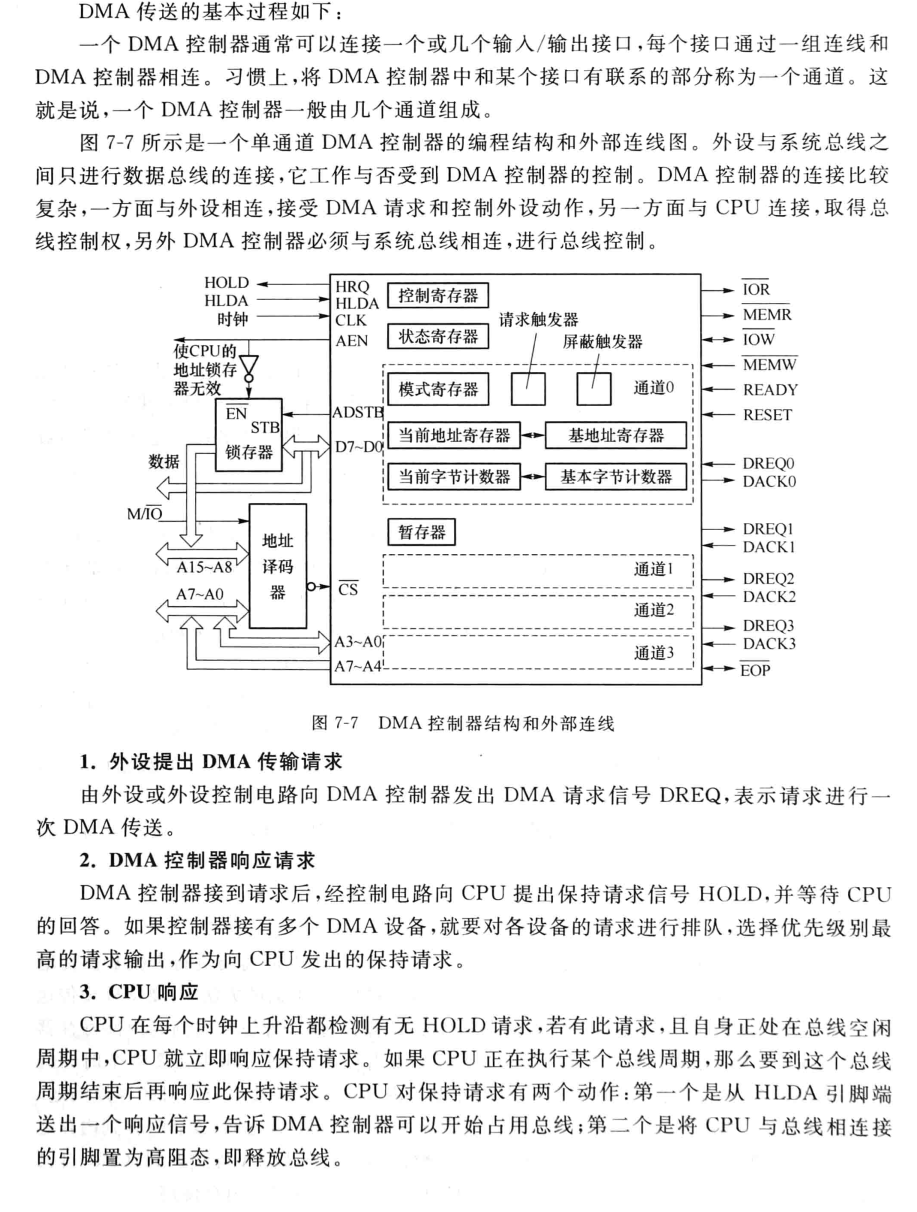
CPU 有转移数据、计算、控制程序转移等很多功能，但其实转移数据（尤其是转移大量数据）是可以不需要 CPU 参与。比如希望外设A 的数据拷贝到外设 B，只要给两种外设提供一条数据通路，再加上一些控制转移的部件就可以完成数据的拷贝。

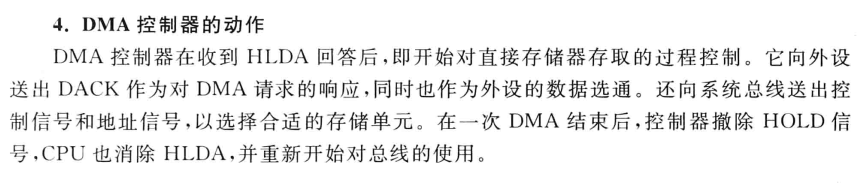
正是基于上述的考虑，大佬们设计了 DMA ，解决数据转移过度消耗CPU资源的问题。

DMA 基本原理：

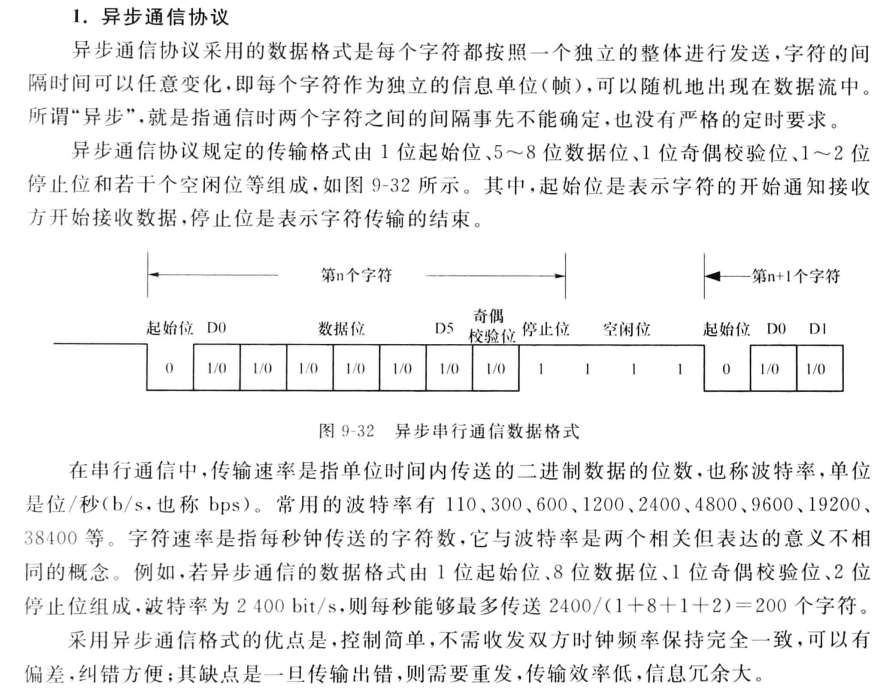
​DMA 是一种允许外围设备（硬件子系统）直接访问系统主内存的机制。也就是说，基于 DMA 访问方式，系统主内存于硬盘或网卡之间的数据传输可以绕开 CPU 的调度。

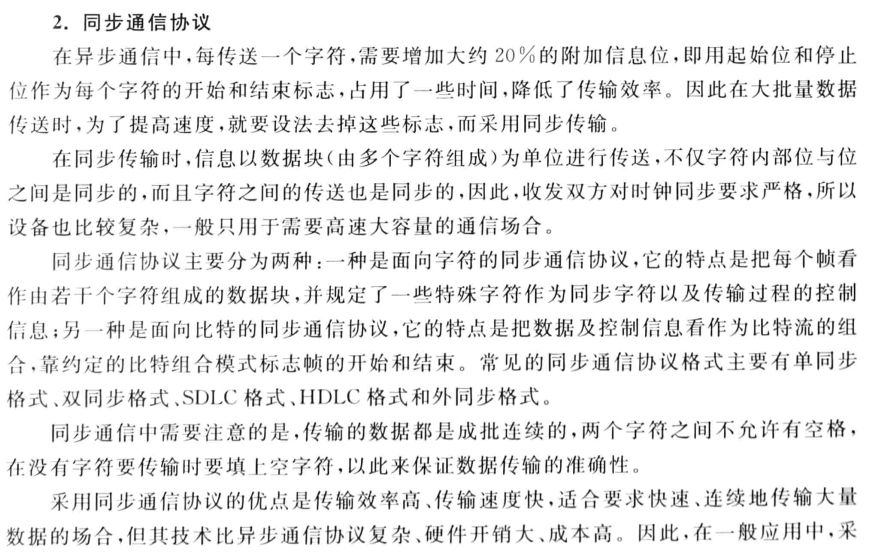
整个数据传输操作在一个 DMA 控制器（DMAC）的控制下进行的，CPU 除了在数据传输开始和结束时做一点处理外（开始和结束时候要做中断处理），在传输过程中 CPU 可以继续进行其他的工作。这样在大部分时间里，CPU 计算和 I/O 操作都处于并行操作，使整个计算机系统的效率大大提高。





**同步异步通信**





用异步通信的比较多。

**Lvds**

LVDS（Low-Voltage Differential Signaling）[低电压差分信号](https://baike.baidu.com/item/%E4%BD%8E%E7%94%B5%E5%8E%8B%E5%B7%AE%E5%88%86%E4%BF%A1%E5%8F%B7/10593300?fromModule=lemma_inlink)，是一种低功耗、低误码率、低串扰和低辐射的差分信号技术，这种传输技术可以达到155Mbps以上，LVDS技术的核心是采用极低的电压摆幅高速差动传输数据，可以实现点对点或一点对多点的连接，其传输介质可以是铜质的PCB连线，也可以是平衡电缆

**flash存储与sram存储器**

FLASH程序存储器存程序，单片机上电后会自动从这里读代码开始运行。 flash是存储芯片的一种，通过特定的程序可以修改里面的数据, **U盘和MP3里用的就是这种存储器.** 目前Flash主要有两种NORFlash和NANDFlash。

NORFlash的读取和我们常见的SDRAM的读取是一样，用户可以直接运行装载在NORFLASH里面的代码，这样可以减少SRAM的容量从而节约了成本。NANDFlash没有采取内存的随机读取技术，它的读取是以一次读取一块的形式来进行的，通常是一次读取512个字节，采用这种技术的Flash比较廉价。用户不能直接运行NANDFlash上的代码，因此好多使用NANDFlash的开发板除了使用NANDFlash以外，还加上了一块小的NORFlash来运行启动代码。

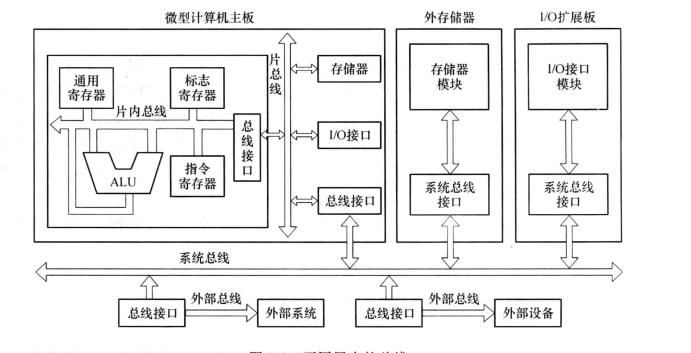
NAND FLASH读取速度与NOR Flash相近，根据接口的不同有所差异；  
NAND FLASH的写入速度比NOR Flash快很多；  
NAND FLASH的擦除速度比NOR Flash快很多；  
NAND FLASH最大擦次数比NOR Flash多；  
NOR Flash支持片上执行，可以在上面直接运行代码；  
NOR Flash软件驱动比NAND FLASH简单；  
NOR Flash可以随机按字节读取数据，NAND FLASH需要按块进行读取。  
大容量下NAND FLASH比NOR Flash成本要低很多，体积也更小

SRAM是跑程序时候暂存临时数据的地方，一般不太大，从128字节到几K字节都有，一掉电数据就没了。

EEPROM是掉电也不丢数据的存储器，一般都用来存设置的。你可以一字节 一字节的把每字节的8位1任意编写成0。但这片一般是按扇区为单位，一擦除就是全成1。

**总线**

数据总线，地址总线，控制总线



总线仲裁：防止因为多个主设备同时传输数据而产生的冲突（总线仲裁器）

需要满足三个要求：

1. 每个时刻只能有一个设备作为总线主控器
2. 现请求者先响应，且一个总线操作周期内不被打断
3. 同一时刻有几个设备发出请求，按优先级排序响应

总线握手：对总线传输进行控制（同步方式，异步方式，半同步方式）

**块 页 扇区**

**块（Block）：**

**块是上层软件中（操作文件时）使用的最小的操作单元。**

就是（操作文件时）一个块 一个块进行操作（块的大小格式化时可以设置【如linux、fatfs等等】）。

块（block），概念来自于**文件系统**，是内核对文件系统数据处理的基本单位，大小为若干个扇区，常见有512B、1KB、4KB等

**扇区：**

**扇区（Sector）**，概念来自于早期磁盘，在硬盘、DVD中还有用，在Nand/SD中已经没意义了，

扇区是块设备本身的特性，大小一般为512的整数倍，

因为历史原因很多时候都向前兼容定义为512，任何块设备硬件对数据处理的基本单位都是扇区。

硬盘的基本访问单位， SATA硬盘一般为512B；

任何块设备硬件对数据处理的基本单位。通常，1个扇区的大小为512byte。（对设备而言）；

**扇区是硬件被操作时软件使用的最小的操作单元。**

就是一个扇区一个扇区进行操作（扇区的大小在存储设备生产时就设计好）。

**扇区和块的关系：**

block由一个或多个sector组成，block是软件（OS、文件系统）中最小的操作单位；

操作系统的虚拟文件系统从硬件设备上读取一个block,实际为从硬件设备读取一个或多个sector。

**block最终要映射到sector上，所以block的大小一般是sector的整数倍**。

**文件和块的关系：**

**块的概念来自于文件系统**；

对于文件管理来说，每个文件对应的多个block可能是不连续的；一个文件至少占用一个块；

**段（Section）:**

概念来自于内核，是内核的内存管理中一个页或者部分页，由若干个连续为块组成。

由若干个相邻的块组成。是Linux内存管理机制中一个内存页或者内存页的一部分。

**页（Page）:**

概念来自于内核，是内核内存映射管理的基本单位。linux内核的页式内存映射名称来源于此。

1. **注意页和块的对象的不同**  
   对**程序**进行**分页**存储  
   对**内存**进行**分块**存储，块是数据存储最小单元

据我的理解：可以理解为指针的传递过程“扇区组合为块和页，页组合为段”，具体组合方式和大小按自己设定的方式用程序进行组合，段地址必须放在寄存器中，这是硬规定。

在操作系统下编程，具体组合方式和大小已经被操作系统设定了，编程人员直接使用就可以了。

如果您自己编操作系统，您也可以自行设定组合方式和大小的。

windows的段、页、块、扇区，是虚拟存储技术（程序控制存储技术）

**可以简单的将扇区和块理解为：扇区是硬件设备传输数据的最小单位，而块是操作系统传输数据的最小单位。一个块通常对应一个或多个相邻的扇区，由于内核将块作为对文件系统操作的最小单位，因此VFS将其看作是单一的数据单元。**

**文件储存在硬盘上，硬盘的最小存储单位叫做"扇区"（Sector）。每个扇区储存512字节（相当于0.5KB）。**

**操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个"块"（block）。这种由多个扇区组成的"块"，是文件存取的最小单位。"块"的大小，最常见的是4KB，即连续八个 sector组成一个 block。**

**文件数据都储存在"块"中，那么很显然，我们还必须找到一个地方储存文件的元信息，比如文件的创建者、文件的创建日期、文件的大小等等。这种储存文件元信息的区域就叫做inode，中文译名为"索引节点"。**

**句柄**

对于操作系统来说，句柄就是指向核心态的某段内存的指针；对于用户来说，就是某段数据的ID。

系统使用句柄来记载数据地址的变更，程序设计中，句柄是一种特殊的智能指针（指向地址的指针），当一个应用程序要引用其他系统（如数据库，操作系统，总线）所管理的内存块或对象。就使用句柄

**函数**

readdir() 返回参数dir 目录流的下个目录进入点

lseek是一个用于改变读写一个文件时读写指针位置的一个系统调用。指针位置可以是绝对的或者相对的。

strncmp函数为字符串比较函数，字符串大小的比较是以ASCII 码表上的顺序来决定，。

函数声明为int strncmp ( const char \* str1, const char \* str2, size\_t n );

功能是把 str1 和 str2 进行比较，最多比较前 **n** 个字节，

若str1与str2的前n个字符相同，则返回0；

若s1大于s2，则返回大于0的值；

若s1 小于s2，则返回小于0的值

Select函数: [https://www.jb51.net/article/246691.htm](%20https://www.jb51.net/article/246691.htm)

Gilbc是提供系统调用和基本函数C库，如open()，malloc(),printyf().所有动态链接库都需要它。

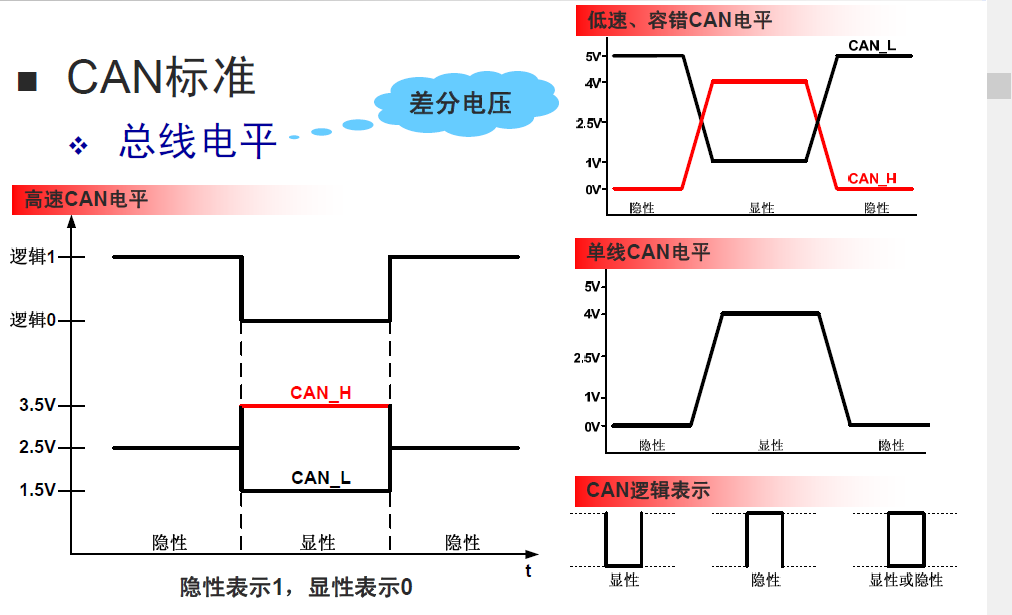
**Can**

载波侦听(CS)：总线上各个节点在发送数据前都要侦听总线的通信状态

如果总线有通信􀃎不发送数据，等待网络空闲。总线空闲􀃎立即发送已经准备好的数据

多路访问(MA)：如果总线空闲，则在同一时刻多个节点可同时访问总线（向总线发送数据）

冲突避免(CA)：节点在发送数据过程中要不停地检测发送的数据，确定是否与其它节点数据发生冲突，并通过非破坏性仲裁机制避免冲突



Can在发送报文过程中进行回读，判断送出的位和回读的位是否一致·

# Shell脚本



## Shell脚本重定向 关于2>&1 &的解释

bash中0，1，2三个数字分别代表STDIN\_FILENO：标准输入（一般是键盘）、STDOUT\_FILENO：标准输出（一般是显示屏，准确的说是用户终端控制台）STDERR\_FILENO标准错误（出错信息输出）

2、输入输出可以[重定向](https://so.csdn.net/so/search?q=%E9%87%8D%E5%AE%9A%E5%90%91&spm=1001.2101.3001.7020)，所谓重定向输入就是在命令中指定具体的输入来源，譬如 cat < test.c 将test.c重定向为cat命令的输入源。输出重定向是指定具体的输出目标以替换默认的标准输出，譬如ls > 1.txt将ls的结果从标准输出重定向为1.txt文本。有时候会看到如 ls >> 1.txt这类的写法，> 和 >> 的区别在于：> 用于新建而 >>用于追加。

3、默认输入只有一个（0，STDIN\_FILENO），而默认输出有两个（标准输出1 STDOUT\_FILENO，标准错误2 STDERR\_FILENO）。因此默认情况下，shell输出的错误信息会被输出到2，而普通输出信息会输出到1。但是某些情况下，我们希望在一个终端下看到所有的信息（包括标准输出信息和错误信息），要怎么办呢？

可以使用2>&1就是用来将标准错误2重定向到标准输出1中的。此处1前面的&就是为了让bash将1解释成标准输出而不是文件1（如果被解释成文件这会创建一个名为1的文件，把2的信息输入到文件1z中）。至于最后一个&，则是让bash在后台执行。

对于command>a 2>&1这条命令，等价于command 1>a 2>&1

可以理解为执行command产生的标准输入重定向到文件a中，标准错误也重定向到文件a中。

那么是否就说command 1>a 2>&1等价于command 1>a 2>a呢。

其实不是，command 1>a 2>&1与command 1>a 2>a还是有区别的，区别就在于前者只打开一次文件a，后者会打开文件两次，并导致stdout被stderr覆盖，可能导致某些输出错误。

&1的含义就可以理解为用标准输出的引用，引用的就是重定向标准输出产生打开的a。从IO效率上来讲，command 1>a 2>&1比command 1>a 2>a的效率更高

https://blog.csdn.net/zhaominpro/article/details/82630528

## Shell脚本创建

1.Vi XXX.sh 创建shell脚本

2.chmod 777 xxx.sh获取控制权，

3. ./ xxx.sh 运行脚本

## Shell脚本比较大小

操作符 含义

-eq 等于（Equal）

-ne 不等于（Not Equal）

-ge 大于或等于（Greater or Equal）

-le 小于或等于（Less or Equal）

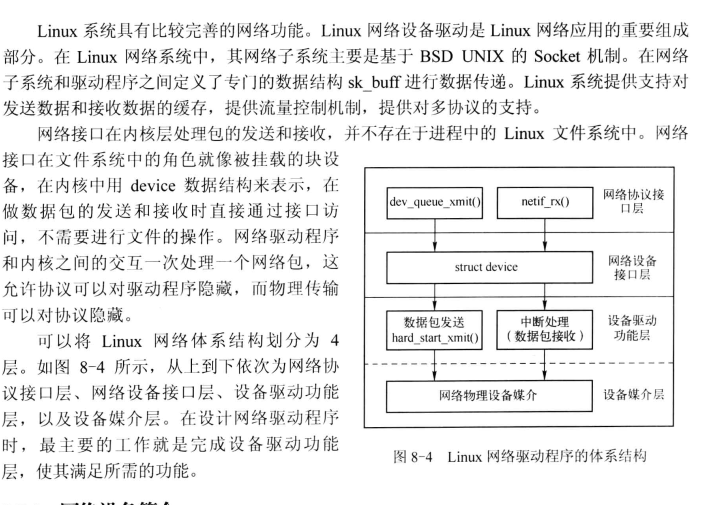
-gt 大于（Greater Than）

-lt 小于（Less Than）

注：

1、在shell中进行比较时，结果为0代表真，为1代表假

# LINUX



## Linux查看版本号

uname -a

Cat /proc/version

Cat /etc/issue

## 挂载

挂载：指的就是将设备文件中的顶级目录连接到Linux根目录下的某一目录，访问此目录就等同于访问设备文件。

Linux 系统使用任何硬件设备，都必须将设备文件与已有目录文件进行挂载。

## LINUX启动

Bootloader引导装载程序（系统加电后运行一段代码）。通过运行该程序可以初始化硬件设备，建立内存空间的映射图，从而将系统的软硬件环境带到一个合适状态，为内核做准备。

针对ARM处理器，常见的Bootloader有U-boot, Redroot和ARMBoot。

系统加电或复位后，所有cpu通常都从某个cpu制造商预先安排的地址上取指令。而基于CPU构建的嵌入式系统通常都有某种类型的固态存储设备（如ROM、EEPROM或Flash等）被映射到这个预先安排的地址上。因此，在系统加电后，CPU首先执行Bootloader程序。

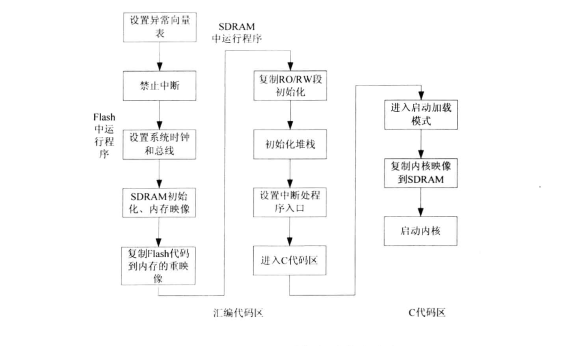
从固态存储设备上启动Bootloader大多是两个阶段的启动过程，启动可分为stage1和stage2两部分。

Stage1依赖于cpu体系结构的代码的初始化，如设备初始化等，使用汇编语言。（为第二阶段执行准备一些必要的硬件环境，并为是stage2准备RAM空间）

1. 硬件设备初始化
2. 为加载Bootloader的stage2准备好一段可用的RAM空间范围。
3. 复制Bootloader的stage2到RAM空间
4. 设置号堆栈
5. 跳转到stage2的C入口点

Stage2通常用C语言来实现，实现复杂功能，代码有可移植性和可读性。

1. 初始化本阶段要使用到的硬件设备
2. 检测系统内存映射 (4gB的物理地址空间并不会全被分配)
3. 将Kernel映像和根文件系统映像从Flash上读取到RAM
4. 将内核设置启动参数
5. 调用内核



## LINUX五大模块

1. 进程调度：控制对cpu的访问
2. 内存管理：允许多个进程安全的共享主内存区域
3. 虚拟文件系统：为所有设备提供统一接口（逻辑文件系统：ext2,fat等和设备驱动程序：为每一种硬件控制器所编写的设备驱动程序模块）
4. 网络接口：提供了对各种网络标准的存取和各种网络硬件的支持
5. 进程通信：IPC支持进程间的各种通信机制

内核配置的过程其实就是对内核进行裁剪的过程，裁减掉一些不必要的功能，节省系统资源，提高运行效率。分为三个步骤：

1. makefile：分布在linux内核源码中的makefile，定义linux内核的编译规则
2. 配置文件（config.in）:给用户提供配置选择的功能
3. 配置工具：包括配置命令编译器和配置用户界面

## 静态链接与动态链接

（1）使用.a 和.lib静态链接库，在运行程序时不需要再挂载dll文件，缺点是导致应用程序比较大，而且失去了动态库的灵活性，发布新版本时要发布新的应用程序才行。

（2）使用lib和dll动态链接库，在应用程序的可执行文件中，存放的不是被调用的函数代码，而是DLL中相应函数代码的地址，使得应用程序比较小，从而节省了存储资源，但是要额外的内存空间用于载入dll中的内容

## GDB调试

程序在发布的时候，同时发布有-g参数和没有-g参数的

在编译阶段会加入某些调试信息

调试信息是在编译的过程中加入到中间文件.o文件的；

（2）release版本

（用户）（发行版本，体积小，没有调试信息）

gcc 默认生成的是release版本。

（3）安装debug（dbg）

命令：sudo apt install dbg

（4）加入调试信息

gcc -o test test.c -g（在编译阶段加的调试信息，非链接阶段）

（5）常用调试命令(重点）

①进入调试：gdb test

②加断点：b+行号

③启动程序：r

④显示代码：l

⑤显示断点信息：info break/info b

⑥删除断点信息：delete 断电编号

⑦单步执行：n

⑧打印：p

⑨自动打印：display

⑩退出：q

⑪进入函数：s

⑫跳入函数：finish

⑬加断点导函数入口处：b+函数名

⑭取消一个display undisplay display的编号

⑮c：继续（continue）

（6）补充命令

①l:显示main函数所在的文件的源代码；

list 文件名:num 显示文件名文件num行上下的源代码；

②b 行号:给指定行添加断点；

b 函数名:给指定函数的第一有效行添加一个断点；

③info break: 显示断点信息; delete 断点号:删除指定断点；

④r(run):运行程序；

⑤n(next):单步执行；

⑥ c(continue):继续执行,直接执行到下一个断点处；

⑦ s:进入将要被调用的函数中执行；

⑧finish:跳出函数；

⑨q:退出调试；

⑩bt:显示函数调用栈；

⑪disable 断点号:将断点设定为无效的,不加断点号,将所有断点设置为无效；

enable 断点号:将断点设定为有效的,不加断点号,将所有断点设置为有效；

⑫p val:打印变量val的值 p &val:打印变量val的地址 p a+b:打印表达式的值；

⑬p arr(数组名):打印数组所有元素的值；

⑭\*parr@len:用指向数组的指针打印数组所有元素的值；

⑮display:自动显示,参数和p命令一样；

⑯info display:显示自动显示信息 undisplay+编号:删除指定的自动显示；

⑰ptype val:显示变量类型。

⑰kill:终止正在进行的程序。

# Git

Git add . 将当前目录下的所有文件添加到缓存区

Git commit -m “说明 ”

git reset --hard 版本号 回退到版本号下的版本

git reset --hard head 恢复当前版本，并删除工作区和缓存区的修改

直接git reset –hard 回退到上一个版本

git push

git config --global core.editor "vim" //用git commit命令没有接-m "说明" 的时候,git会尝试打开一个编辑器让你输入更改说明,这个时候就会打开vim编辑器,也是就我们git config --global core.editor vim命令指定的文本编辑器,如果将这条命令换成git config --global core.editor gedit,使用git commit的时候会弹出geditor编辑器让我们输入更改说明.

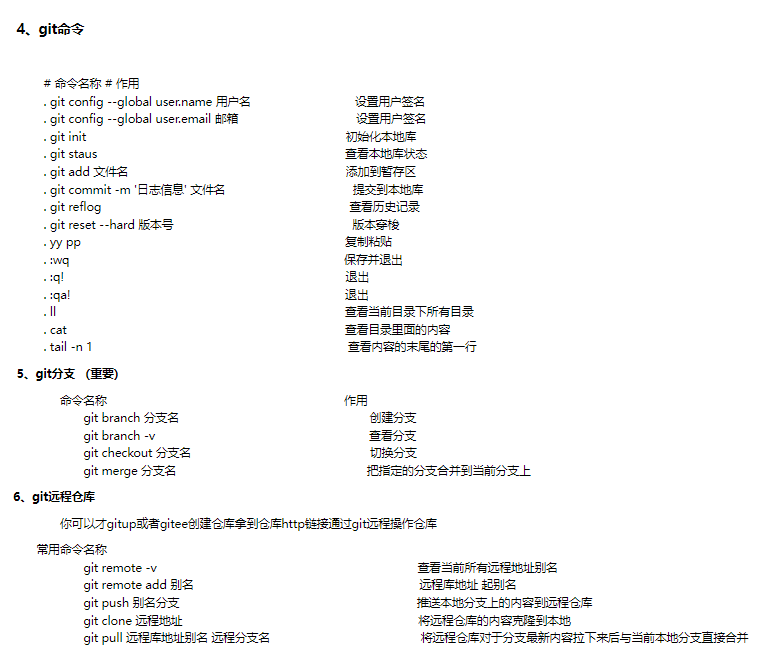
global config –global user.name “liudongyue”

global config –global user.email “邮箱”

如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global 选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

Git config --list 查看config的配置（包括用户邮箱 用户名等）

git rm/git mv 将文件从暂存区和工作区中删除/ 移动或重命名工作区文



# 操作系统

## 进程控制

程序是存储在磁盘上包括可执行机器指令和数据的静态实体

进程或者任务是处于活动状态的计算机程序。

每个进程都有创建时会被分配一个数据结构体，该结构体被称为进程控制块或进程描述符（pcb 是一个task\_struct结构体，每个进程会被分配一个）,包括进程ID等信息。

Linux进程包括三大块：代码块，数据块，堆栈块（存放动态分配到内存变量）。

Linux进程有五个状态：可运行，可中断睡眠，不可中断睡眠，暂停，僵死。

进程通信：

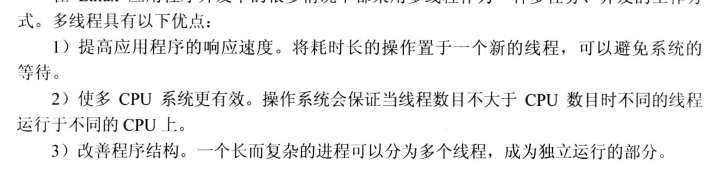
1. 管道:半双工 运行一个进程和另一个与他有共同祖先的进程之间进行通信
2. 有名管道：半双工 同上以为，还允许无亲缘关系进程间的通信
3. 消息队列： 存放在内核中的消息链表。克服信号传递信息量少，缓冲区大小受限等缺点
4. 信号量：计数器，用于同一进程中各个线程间的信息交互，常作为锁机制。本身不具备数据交换能力，而是控制其他通信资源（文件，外围设备）来实现进程间通信。
5. 共享内存：多个进程共享一块内存空间
6. 信号
7. 套接字：可用于不同机器间的进程通信

管道一旦创建成功，就可以作为一般文件使用，对一般文件进行操作的I/O函数open, read, write也适用

## 线程

[**https://baijiahao.baidu.com/s?id=1687308494061329777&wfr=spider&for=pc**](https://baijiahao.baidu.com/s?id=1687308494061329777&wfr=spider&for=pc)

**进程是操作系统分配资源的单位，线程是调度的基本单位，线程之间共享进程资源**”。



**线程运行的本质其实就是函数的执行，线程的出现是为了更好的支持多处理器，减少（进程/线程）上下文切换的开销。**

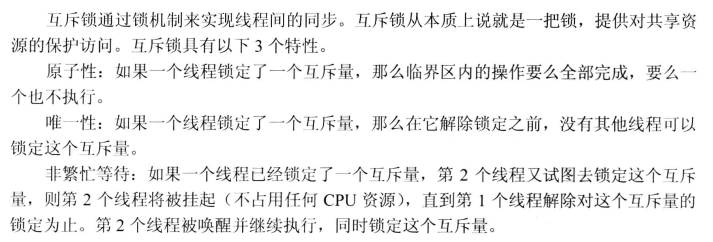
**所属线程的线程号、栈区、程序计数器、栈指针以及函数运行使用的寄存器是线程私有的。 剩下的（堆区，代码区，数据区，栈区（**栈区是线程的私有数据，但由于栈区没有添加任何保护机制，一个线程的栈区对其它线程是可以见的，也就是说我们可以修改属于任何一个线程的栈区。**））都是线程间共享资源。**

以上这些信息有一个统一的名字，就是线程上下文，thread context。

线程私有数据采用一键多值的技术

**线程私有数据实现的主要思想是：** 在分配线程私有数据之前，创建与该数据相关联的键，这个键可以被进程中的所有线程使用，但**每个线程把这个键与不同的线程私有数据地址进行关联**，需要说明的是每个系统支持有限数量的线程特定数据元素

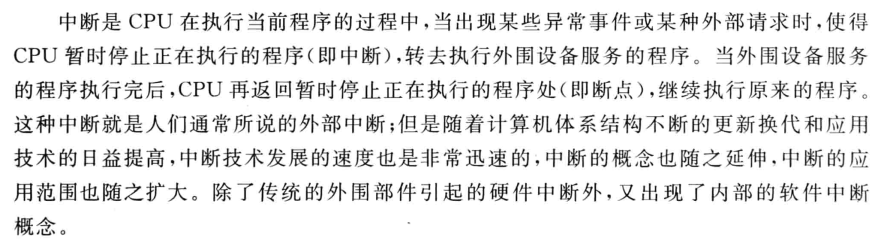
线程同步机制：互斥锁（原子性，唯一性，非繁忙等待），条件变量，信号量



信号量：是一种特殊的变量，只能取正整数，并只能采取P操作（代表等待、关操作）、V操作（代表信号、开操作）

## 中断

为了提高输入/输出数据的吞吐率，加快运算速度便产生中断技术。



中断矢量表中存在的是中断服务程序的入口地址，

中断处理过程：

1. 中断请求
2. 中断响应
3. 中断处理
4. 中断返回

## 阻塞和非阻塞

为了完成一个功能,发起一个调用,如果不具备条件的话则一直等待,直到具备条件则完成

阻塞

为了完成一个功能,发起一个调用,**如果不具备条件的话则一直等待,直到具备条件则完成** 当前线程会被挂起，并在得到结果之后返回

非阻塞

为了完成一个功能,发起一个调用,具备条件直接输出,不具备条件直接报错返回

对于非阻塞的使用必须使用循环进行调用

如果不能立刻得到结果，则该调用者不会阻塞当前线程

区别

其实就相当于在捕捉一个子进程退出的时候，阻塞则会一直等待，直到这个子进程退出，返回对应的值，而非阻塞，如果刚好捕捉到子进程的退出则直接输出 ，如果没有捕捉到,也不进行等待,直接输出报错!

# Linux常用命令：

## Linux查看版本号和软链接

uname -a

Cat /proc/version

Cat /etc/issue

Ln -s实现软连接

## du查看文件大小

du -sh <文件名>

du -sh apps //查看文件夹得大小 以M返回

ls -lh(不准)

ls -l | grep "^-" | wc -l //统计当前目录下文件的个数（不包含目录或子文件夹）

ls -lR | grep "^-" | wc -l //统计当前目录下文件的个数（包含子目录中的文件）

ls -l | grep "^d" | wc -l //查看当前路径下文件夹（目录）的个数（不包含子目录）lR 包含子目录中的文件

## Linu上ssh安装

ssh方便一个开发小组中人员登录一台服务器，从事代码的编写、编译、运行。方便代码的共享及管理。ssh是一种安全协议，主要用于给远程登录会话数据进行加密，保证数据传输的安全。

如果你只是想登陆别的机器的SSH只需要安装openssh-client（ubuntu有默认安装，如果没有则sudo apt-get install openssh-client），如果要使本机开放SSH服务就需要安装openssh-server

sudo apt-get install openssh-serve

然后确认sshserver是否启动了：

ps -e |grep ssh

如果看到sshd那说明ssh-server已经启动了。

如果没有则可以这样启动：sudo /etc/init.d/ssh start

vi /etc/ssh/sshd\_config

    修改 PermitRootLogin yes

    /etc/init.d/ssh restart

    即可使用sftp和ssh工具连接到ubuntu

## Cp的目录下还有子目录时

递归去cp

Cp -r apps ldy

time cp -r apps ldy

## Grep过滤查询

Tail -f XX |Grep -P “XXX|XXX|XXX” –-color; 或者 - -always;可以同时匹配多个关键字,并且高亮显示

Tail -f XX |Grep -E “XXX|XXX|XXX” –-color; 或者 - -always;

grep -E 和 -P 区别, 推荐 -P 参数, 因为 -E 是不完整的表达式, 而 -P 是完整的表达式.

**ps -ef | grep --color=always 6379**

|  |  |
| --- | --- |
| c | 仅列出文件中包含模式的行数。 |
| -i | 忽略模式中的字母大小写。 |
| -l | 列出带有匹配行的文件名。 |
| -n | 在每一行的最前面列出行号。 |
| -v | 列出没有匹配模式的行。 |
| -w | 把表达式当做一个完整的单字符来搜寻，忽略那些部分匹配的行。 |

grep -c CLERK emp.data 在emp文件里职位为 CLERK 的员工的人数

## more以页的形式显示文本文件的内容

more 命令可以分页显示文本文件的内容，使用者可以逐页阅读文件中内容，此命令的基本格式如下：

[root@localhost ~]# more [选项] 文件名

|  |  |
| --- | --- |
| -f | 计算行数时，以实际的行数，而不是自动换行过后的行数。 |
| -p | 不以卷动的方式显示每一页，而是先清除屏幕后再显示内容。 |
| -c | 跟 -p 选项相似，不同的是先显示内容再清除其他旧资料。 |
| -s | 当遇到有连续两行以上的空白行时，就替换为一行的空白行。 |
| -u | 不显示下引号（根据环境变量 TERM 指定的终端而有所不同）。 |
| +n | 从第 n 行开始显示文件内容，n 代表数字。 |
| -n | 一次显示的行数，n 代表数字。 |