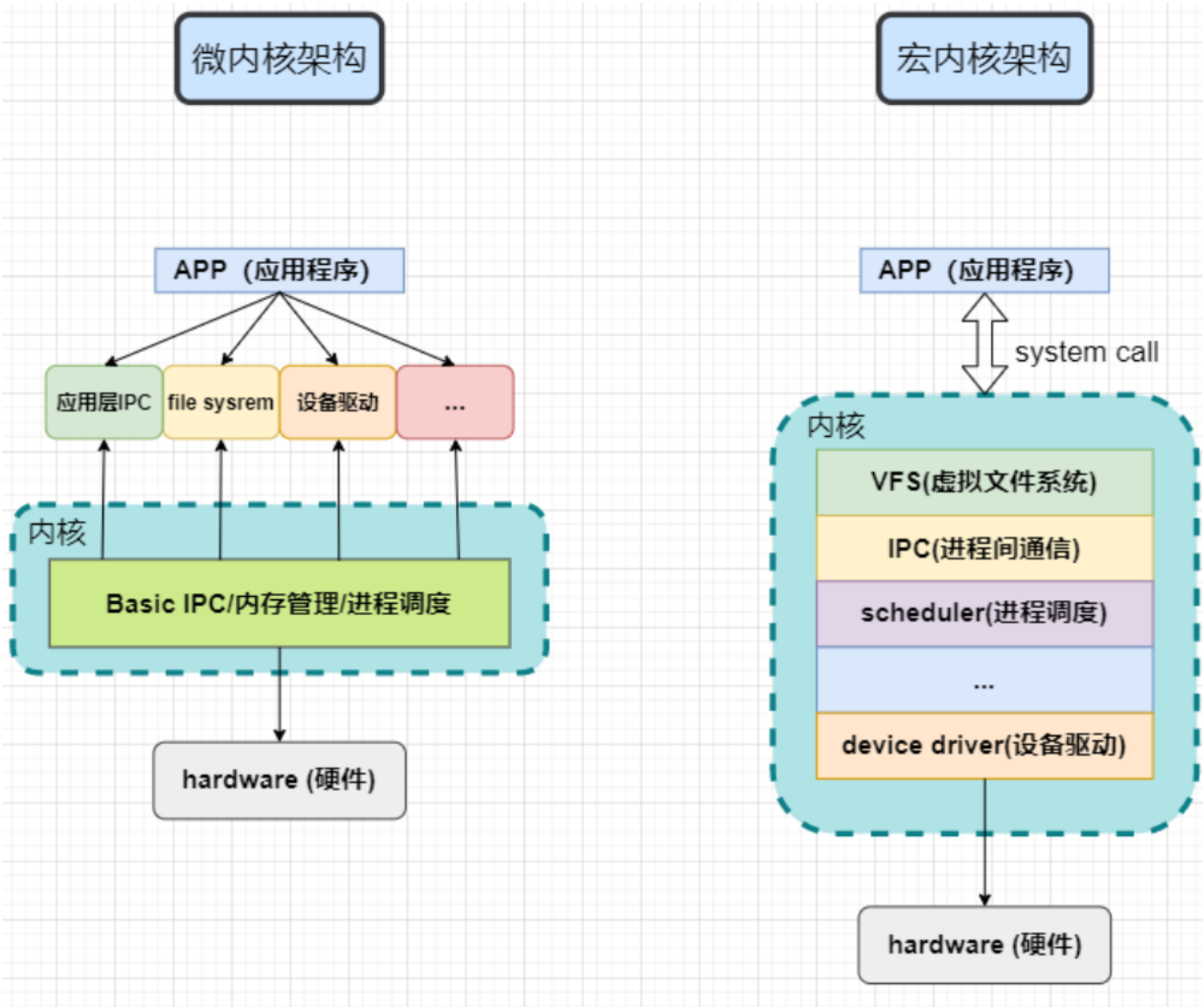
00-linux笔记

开源地址

1. Shared Libraries: Understanding Dynamic Loading

1. 内核详解



2. Linux源设置

2.1 源地址

3.1 Ubuntu 中国官方源

archive 源：http://cn.archive.ubuntu.com/ubuntu/

ports　源：http://cn.ports.ubuntu.com/ubuntu-ports/

Ubuntu 中国官方源好像用的是阿里云，换源说明请参考阿里源和中科大源的换源说明。

3.2 阿里源

archive 源：https://mirrors.aliyun.com/ubuntu/

ports　源：https://mirrors.aliyun.com/ubuntu-ports/

换 源介 绍：https://developer.aliyun.com/mirror/ubuntu（ archive 源 ）

阿里云只有 archive 源的换源说明，其实 archive 源和 ports 源的换源步骤都是一样的，只是源地址不一样。

3.3 清华源

archive 源：https://mirrors.tuna.tsinghua.edu.cn/ubuntu/

ports 源：https://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/

换 源介 绍：https://mirror.tuna.tsinghua.edu.cn/help/ubuntu/（ archive 源 ）

清华大学也只有 archive 源的换源说明，但说明下面提示了ARM 和 PowerPC架构使用 ubuntu-ports 源。

3.4 中科大源

archive 源：https://mirrors.ustc.edu.cn/ubuntu/

ports 源：https://mirrors.ustc.edu.cn/ubuntu-ports/

换 源介 绍：https://mirrors.ustc.edu.cn/help/ubuntu.html（ archive 源 ）

https://mirrors.ustc.edu.cn/help/ubuntu-ports.html（ ports 源 ）

中国科学技术大学开源镜像站对两种源的换源过程是介绍得非常清楚的。

https://blog.csdn.net/xiangxianghehe/article/details/122856771

2.2 更改文件位置

ubuntu在/etc/apt/sources.list

CentOS 8在/etc/yum.repos.d/

2.3 修改源

```
sudo sed -i -e "s|mirrorlist=|#mirrorlist=|g" /etc/yum.repos.d/CentOS-*
sudo sed -i -e
"s|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g"
/etc/yum.repos.d/CentOS-*
```

2.4 更新

```
yum makecache
```

2.5 配置文件示例

```
sources.list 20.4

#添加阿里源
deb http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe multiverse
#添加清华源
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main restricted universe multiverse
```

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-security main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-security main restricted universe multiverse multiverse

sources.list 18.4

#阿里源
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse

#清华源
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic main restricted universe multiverse
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-updates main restricted universe multiverse
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-backports main restricted universe multiverse
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-security main restricted universe multiverse
```

```
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ bionic-proposed main restricted universe multiverse
deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/ bionic stable
# deb-src [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/ bionic stable
```

3. 环境变量

3.1 设置多个环境变量

```
export LD_LIBRARY_PATH=/the/path/you/want/set:/home/data
```

3.2 查看设置

```
echo $LD_LIBRARY_PATH
```

3.3 清除设置

```
unset LD_LIBRARY_PATH
```

3.4 配置永久环境变量

```
export MY_VARIABLE=value
```

临时环境变量，只在当前 shell 会话中有效，一旦退出该会话，这些变量就会失效。

1. .bash_profile:

   `.bash_profile` 是用于登录 Bash shell 时执行的配置文件，打开一个新的终端窗口时，`.bash_profile` 文件会被执行，并用于设置用户的个性化环境。通常，`.bash_profile` 中会设置一些全局的环境变量、路径、别名等，这些设置会对整个会话持续生效。

2. .bashrc:

   `.bashrc` 是用于每次打开新的 Bash shell 时执行的配置文件。当你在已经登录的 Bash shell 中打开一个新的子 shell 时，`.bashrc` 文件会被执行。`.bashrc` 文件通常包含了一些与 shell 相关的设置，如别名、函数、命令行提示等。这些设置只对当前的 Bash shell 会话生效。

`.bash_profile` 和 `.bashrc` 文件通常位于用户的家目录（home directory）下

4. 用户更改

4.1 查看用户who

查看登录账号命令

```
li_jk@ljkpc:~$ who
li_jk    :0           2023-07-01 22:25 (:0)
li_jk    pts/3        2023-07-02 11:26 (192.168.20.139)
```

":0" 表示图形用户界面（GUI）会话。这通常是指本地登录的桌面环境，例如 X Window System。当用户通过图形登录界面（如登录管理器）登录到系统时，会创建一个新的图形会话，并分配一个唯一的终端 ID，通常以 ":0" 结尾。

"pts/3" 表示伪终端会话。这是通过终端仿真器（如终端窗口、SSH 连接等）远程登录到系统的会话。每当用户通过终端仿真器登录时，会为其创建一个伪终端（pseudo-terminal，简写为 pts）并分配一个唯一的终端 ID，后面跟着一个数字。

4.2 更改用户名

3.1 添加用户

adduser +用户名

adduser +用户名

创建一个用户

下面会接下设置

3.2 useradd +用户名

useradd +用户名 ，它并没有在/home目录下创建同名文件夹，也没有创建密码，因此利用这个用户登录系统，是登录不了的，为了避免这样的情况出现，可以用 （useradd -m +用户名）的方式创建，它会在/home目录下创建同名文件夹，然后利用（ passwd + 用户名）为指定的用户名设置密码

3.3 删除用户

只需使用一个简单的命令“userdel 用户名”即可。不过最好将它留在系统上的文件也删除掉，你可以使用“userdel -r 用户名”来实现这一目的

3.4 添加用户提升权限

修改/etc/sudoers

root ALL=(ALL) ALL

usr ALL=(ALL) ALL %usr我们创建的用户，照着root那行超一下就行

注意一定要先sudo su切换为root权限再切换权限

chmod 777 /etc/sudoers

再修改sudoers，否则无法普通用户sudoers权限后无法再提升权限，也切换root用户，系统就废了

3.5 创建Ubuntu 时

姓名：登录时显示的名字

计算机名:网络中计算机的名字，跟物理硬件网卡有关系

用户名:linux的用户，shell终端显示的

用户名@计算机名:~$

$普通用户

#root用户

4.3 修改用户密码

（1）进入Ubuntu，打开终端，输入：sudo su 转为root用户；

（2）输入：sudo passwd user(user 是对应的用户名)；

（3）输入新密码，确认密码；

(4) 修改密码成功，重启，输入新密码进入Ubuntu；

4.4 修改用户名

（1） 进入ubuntu，打开一个终端，输入：sudo su 转为root用户；

（2）gedit /etc/passwd ，找到代表你的那一行，修改用户名为新用户名；

（3）gedit /etc/shadow ，找到代表你的那一行，修改用户名为新用户名；

（4）gedit /etc/group ，你应该发现你的用户名在很多个组中，全部修改；

（5）修改完毕！

4.5 设置默认登录账户 （没有成功）

1、启用 root 帐号

```
$ sudo passwd root
输入新的 UNIX 密码：
$ sudo passwd --unlock root
passwd：密码过期信息已更改。
```

2、重启后以 root 身份登录(重启用于关闭 pi 打开的程序)

3、修改用户名 pi 为 upall

```
usermod -l upall pi
```

4、修改组名 pi 为 upall

```
groupmod -n upall pi
```

5、更改 pi 的家目录 为 upall 的家目录

```
mv /home/pi /home/upall
```

6、修改 /etc/passwd 中 upall 用户的家目录地址

```
usermod -d /home/upall upall
```

7、允许 upall 用户使用 sudo 命令需要编辑 /etc/sudoer 文件，将末尾的

```
pi ALL=(ALL) NOPASSWD: ALL
改为:
upall ALL=(ALL) NOPASSWD: ALL
```

8、重新锁定 root 用户

```
passwd -l root
```

9、以新终端以 upall 身份登录并测试sudo，如果正常的话退出上一个窗口，操作即完成。

4.6 修改主机名

修改主机名

1、修改配置文件

修改hostname文件

输入：sudo vim /etc/hostname ， 把旧主机名修改为新主机名；

修改hosts文件

输入：sudo vim /etc/hosts ，把旧主机名修改为新主机名；

2、命令修改

```
sudo hostnamectl set-hostname newname
sudo reboot
```

4.7 用户配置信息

/etc/shadow 是一个位于 Linux 系统上的重要系统文件，用于存储用户帐户的加密密码哈希和其他相关安全信息。这个文件通常只能由系统管理员访问，并包含以下信息：

用户名：	用户的登录名。
密码哈希：	用户的密码哈希值，通常是加密的，以确保密码的安全性。
密码策略：	包括密码最长有效期、最短有效期、密码过期警告期等信息。
帐户锁定信息：	如账户是否被锁定、锁定的日期等。
密码过期日期：	密码过期的具体日期。
最近修改密码日期：	用户最后一次修改密码的日期。
密码历史：	以前使用的密码的记录，以防止用户在短时间内多次使用相同的密码。
用户 ID 和组 ID：	与用户相关联的数字标识符。
其它信息：	可能包括用户的全名、联系信息等。

5. 系统查询

5.1 操作系统基本查询

CPU查询

```
cat /proc/cpuinfo
```

系统查询

```
uname -a
cat /proc/version
```

位数查询

```
1、getconf LONG_BIT 或 getconf WORD_BIT
2、file /bin/ls
3、lsb_release  -a
```

查看系统什么类型linux

```
cat /etc/issue
ubuntu  centos  deep debian
```

其他查询

cpu使用率查询

```
/proc/stat
```

ram查询

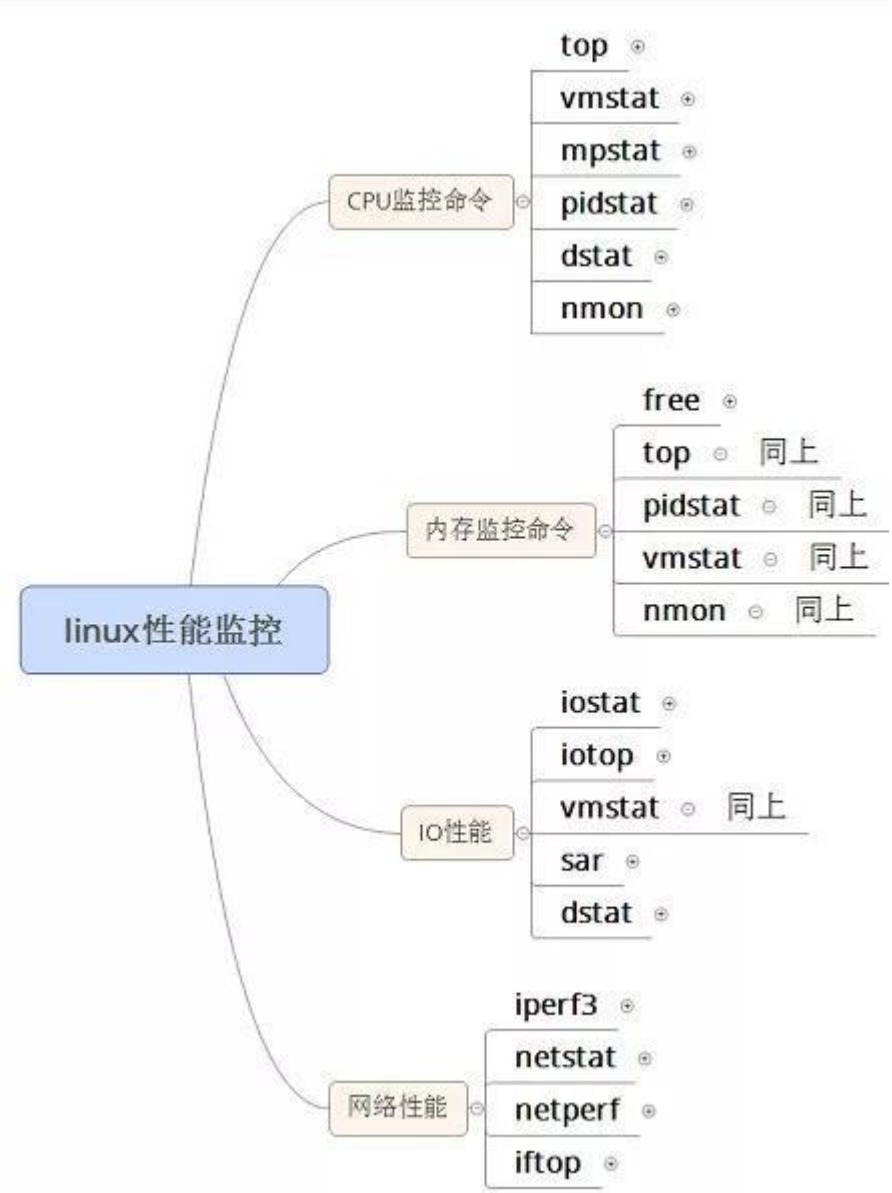
```
/proc/meminfo

free -h
```

rom查询

```
/etc/mtab
```

5.2 性能查询



-n 网络里显示ip而不是域名

-h 内存里安装最大单位GB等来显示，不是KB或者B单位

5.2.1 内存查取命令

/proc 文件

/proc/meminfo	机器的内存使用信息
/proc/pid/maps	pid为进程号，显示当前进程所占用的虚拟地址
/proc/pid/statm	进程所占用的内存
/proc/pid/status	某个进程占用内存信息，还包括进程IDs、信号等信息
/proc/pid/smaps	查看进程的虚拟内存空间的分布情况
/proc/pid/status	中所保存的信息除了内存信息，还包括进程IDs、信号等信息，此处暂时只介绍内存相关的信息。

/proc/meminfo 字段详细解释

字段	说明
VmPeak	进程所使用的虚拟内存的峰值
VmSize	进程当前使用的虚拟内存的大小
VmLck	已经锁住的物理内存的大小（锁住的物理内存不能交换到硬盘）
VmHWM	进程所使用的物理内存的峰值
VmRSS	进程当前使用的物理内存的大小
VmData	进程占用的数据段大小
VmStk	进程占用的栈大小
VmExe	进程占用的代码段大小（不包括库）
VmLib	进程所加载的动态库所占用的内存大小（可能与其它进程共享）
VmPTE	进程占用的页表大小（交换表项数量）
VmSwap	进程所使用的交换区的大小

```
cat /proc/pid/status
查看进程占用内存 pid为进程号
VmHWM:            47940kB  RSS峰值。
VmRSS:            47940kB RSS实际使用量=RSSAnon+RssFile+RssShmem。
RssAnon:          38700 kB
RssFile:          9240 kB RssFile是库代码映射，是多个进程公用，所以如果以VmRSS的值作为参考，获得的内存会比较大。
```

5.2.2 top命令

top –d 1	时间间隔1s
top -p pid	观察单独进程

字段参数含义

```
PID    USER    PR  NI  VIRT    RES   SHR   S  %CPU  %MEM     TIME+  COMMAND
21829  root    20   0  0       0     0     S   0.7  0.6   129:53.91  java
22559  root    20   0  158920  5628  4268  S   0.3  9.2   139:42.81  java
22598  root    20   0  162112  2208  1540  S   0.3  0.1   0:04.68    fluentd
```

PID	进程id
USER	进程所有者的用户名
PR	优先级
NI	nice值，负值表示高优先级，正值表示低优先级
VIRT	进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES
RES	进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA
SHR	共享内存大小，单位kb
S	进程状态。D=不可中断的睡眠状态 R=运行 S=睡眠 T=跟踪/停止 Z=僵尸进程
%CPU	上次更新到现在的CPU时间占用百分比
%MEM	进程使用的物理内存百分比
TIME+	进程使用的CPU时间总计，单位1/100秒
COMMAND	命令名/命令行

5.2.3 free命令

free -h

实际内存查询

Mem	行是物理内存使用情况
Swap	行是虚拟内存作用情况
used	列为已分配内存
free	列为未分配的内存

total	列为物理内存总量
shared	列为共享内存
available	列为可用内存

5.2.4 vmstat命令

vmstat

```
[root@jingyang /]# vmstat  -S m  1 5
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0    879     48     53    0    0    18     4   16   17  0  0 99  1  0
 0  0      0    879     48     53    0    0     0     0   12   10  0  0 100  0  0
 0  0      0    879     48     53    0    0     0     0   13   12  0  0 100  0  0
 0  0      0    879     48     53    0    0     0     0   12   13  0  0 100  0  0
 0  0      0    879     48     53    0    0     0     0   13   12  0  0 100  0  0
```

Procs	R:等待被执行的进程数，即表示运行和等待CPU时间片的进程数 B:排队的进程数，即等待资源的进程数
Memory	Swap : 虚拟内存，切换到虚拟内存的内存大小 Free: 空闲的物理内存大小 Buff: 缓冲区大小 Cache: 缓存大小
Swap	Si:磁盘写入虚拟内存，即由内存进入到虚拟内存的大小。 So:虚拟内存写入磁盘，即由虚拟内存进入到磁盘的大小。
Io	Bi:由块设备读入的数据总量，读磁盘 Bo:由块设备写入的数据总量，写磁盘
System	In: 每秒设备中断数 Cs:每秒上下文切换的次数
Cpu	Us:用户进程消耗cpu百分比 Sy:内核进程消耗cpu百分比 Id:cpu处于空闲状态的时间百分比 Wa：Io等待cpu所占时间的百分比

5.2.5 lsof命令

lsof

- -a： 列出打开文件存在的进程

- -c<进程名>：列出指定进程所打开的文件
- -g：列出GID号进程详情
- -d<文件号>：列出占用该文件号的进程
- +d<目录>：列出目录下被打开的文件
- +D<目录>：递归列出目录下被打开的文件
- -n<目录>：列出使用NFS的文件
- -i<条件>：列出符合条件的进程。
- -p<进程号>：列出指定进程号所打开的文件
- -u 后面跟username：列出该用户相关进程所打开文件
- -U：仅列出系统socket文件类型
- -h：显示帮助信息
- -v：显示版本信息

COMMAND：	进程的名称
PID：	进程标识符
USER：	进程所有者
FD：	文件描述符，应用程序通过文件描述符识别该文件。如cwd、txt等
TYPE：	文件类型，如DIR、REG等
DEVICE：	指定磁盘的名称
SIZE：	文件的大小
NODE：	索引节点（文件在磁盘上的标识）
NAME：	打开文件的确切名称

5.2.6 iostat命令

iostat

Iostat是对系统磁盘IO操作进行监控，它的输出主要显示磁盘的读写操作的统计信息。同时给出cpu的使用情况。

5.2.7 mpstat命令

mpstat

5.2.8 watch命令

-d	高亮显示变动
-n	周期（秒）

5.2.9 strace命令

`strace` 是一个用于跟踪进程系统调用的工具。它可以捕获并记录进程执行的系统调用以及接收和发送的信号。

- -p：跟踪指定的进程。
- -f：跟踪由fork子进程系统调用。
- -F：尝试跟踪vfork子进程系统调吸入，与-f同时出现时, vfork不被跟踪。
- -o filename：默认strace将结果输出到stdout。通过-o可以将输出写入到filename文件中。
- -ff：常与-o选项一起使用，不同进程(子进程)产生的系统调用输出到filename.PID文
- -r：打印每一个系统调用的相对时间。
- -t：在输出中的每一行前加上时间信息。-tt 时间确定到微秒级。还可以使用-ttt打印相对时间。
- -v：输出所有系统调用。默认情况下，一些频繁调用的系统调用不会输出。
- -s：指定每一行输出字符串的长度,默认是32。文件名一直全部输出。
- -c：统计每种系统调用所执行的时间，调用次数，出错次数。
- -e expr：输出过滤器，通过表达式，可以过滤出掉你不想要输出。
- -d：输出strace关于标准错误的调试信息。
- -h：输出简要的帮助信息。
- -i：输出系统调用的入口指针。
- -q：禁止输出关于脱离的消息。
- -tt：在输出中的每一行前加上时间信息,微秒级。
- -T：显示每一调用所耗的时间。
- -V：输出strace的版本信息。
- -x：以十六进制形式输出非标准字符串。

- -xx：所有字符串以十六进制形式输出。

5.2.10 手动配置交换空间

```
#!/bin/bash

swap_size=8

# 禁用当前的 Swap 文件
echo "禁用当前的 Swap 文件"
sudo swapoff /swapfile

# 增加 Swap 文件的大小
echo "加 Swap 文件的大小: ${swap_size}G"
sudo fallocate -l ${swap_size}G /swapfile
#sudo dd if=/dev/zero of=/swapfile bs=1G count=$swap_size

# 设置 Swap 文件的权限
echo "设置 Swap 文件的权限"
sudo chmod 600 /swapfile

# 制作 Swap 空间
echo "制作 Swap 空间"
sudo mkswap /swapfile

# 激活 Swap 文件
echo "激活 Swap 文件"
sudo swapon /swapfile

# 确认 Swap 文件激活
echo "确认 Swap 文件激活"
sudo swapon --show
```

5.3 磁盘命令

linux下磁盘或存储设备是不能被直接访问的

所有存储设备上的具体文件内容需要将存储设备挂载到操作系统的某一个文件夹下，通过该文件夹来访问设备存储的文件内容或上传到该存储设备上

linux下硬盘命名从sda开始，依次是sdb、sdc、sdd .....

硬盘下分区的命名从sda1开始，依次是sda2、sda3 .....

5.3.1 挂载信息

5.3.1.1 mount命令

类 Unix 系统中挂载本地或远程的文件系统

将某个文件分区挂载到文件夹

格式：mount 分区路径 文件夹路径/文件夹名(建议文件夹路径从根路径开始)

注意：挂载的文件夹最好是一个空文件夹

使用mount挂载，该挂载的分区仅在电脑系统关机重启之前有效

如果要实现自动挂载磁盘，可以在 /etc/fstab 文件中添加一行

```
/dev/sdb1  /usr/cipan3 ext3 default 0 0
分区路径    挂载文件夹路径   文件格式
```

示例

```
磁盘挂载
在 /mnt 目录下新建一个 home,并将我们的 /dev/sda2 挂载上去。
cd /mnt
sudo mkdir home
sudo mount /dev/sda2 /mnt/home

卸载
umount /dev/sdb1
```

5.3.1.2 findmnt命令

运行信息

展示出了目标挂载点（TARGET）、源设备（SOURCE）、文件系统类型（FSTYPE）以及相关的挂载选项（OPTIONS）。

例如文件系统是否是可读可写或者只读的。根（/）文件系统的类型是xfs。

分区时，输入 p 代表主分区，输入 e 代表逻辑分区

分区编号自己设置，占用空间大小使用默认值表示将整个硬盘空间都分配给这个分区

注：一块新的硬盘必须进行分区和格式化才能够使用，只有格式化了才能确定该硬盘上面的文件存储格式

5.3.1.3 lsblk命令

显示已挂载文件系统的信息

5.3.1.4 fdisk 命令

对新的硬盘进行分区和格式化

fdisk -l 命令，可以列出所有分区

用法：fdisk 分区路径 ： 进入分区

Command(m for help)： n --输入 n 进行分区

5.3.1.5 dd命令

复制文件： `dd` 命令可以用来从一个文件或设备复制数据到另一个文件或设备。例如，将一个磁盘镜像写入到另一块磁盘：

```
1 bashCopy code
2 dd if=input_file of=output_file
```

这将从 `input_file` 读取数据，并将其写入到 `output_file` 。

创建空文件： `dd` 命令也可以用来创建指定大小的空文件。例如，创建一个大小为1GB的空文件：

```
1 bashCopy code
2 dd if=/dev/zero of=empty_file bs=1G count=1
```

这将从 `/dev/zero` 设备读取数据，并将其写入到 `empty_file` 中， `bs` 参数指定了块大小， `count` 参数指定了块数量。

转换文件格式： `dd` 命令可以用来执行各种数据格式的转换，如大/小端字节序的转换、ASCII到EBCDIC的转换等。

备份和恢复： `dd` 命令可以用来创建磁盘的完整备份和恢复，包括操作系统、文件系统和数据。

5.3.1.6 syn命令

sync 命令是一个Linux/Unix系统中的实用工具，用于将内存中的数据同步到磁盘中，确保文件系统的一致性。它会强制将所有未写入磁盘的缓冲区数据刷新到磁盘中，以防止数据丢失或损坏。

5.3.2 硬盘大小

5.3.2.1 df -h命令：查看当前磁盘和磁盘分区的使用情况

用于查看文件系统的整体磁盘使用情况，提供文件系统级别的信息。

```
pi@raspberrypi:~/test $ df -h
文件系统        容量  已用  可用 已用% 挂载点
/dev/root        59G   24G   33G   42% /
devtmpfs        1.8G     0  1.8G    0% /dev
tmpfs           1.9G     0  1.9G    0% /dev/shm
tmpfs           768M  1.2M  767M    1% /run
tmpfs           5.0M  4.0K  5.0M    1% /run/lock
/dev/mmcblk0p1  253M   50M  203M   20% /boot
tmpfs           384M   28K  384M    1% /run/user/1000
pi@raspberrypi:~/test $ du -sh
1.9M    .
pi@raspberrypi:~/test $ du -h
40K     ./test
48K     ./open
1.9M    .
```

依次是：分区 分区的大小 已使用大小 空闲大小 使用率 挂载的文件夹

```
1 df -a：查看所有的磁盘使用，包含隐藏的内容
2 df -h：以最大单位显示分区大小，已使用分区大小，空闲大小的磁盘使用
```

5.3.2.2 du *命令：查看文件夹占用的空间大小

当前目录下各个文件详细大小

用于计算特定文件或目录的磁盘使用量，提供文件或目录级别的详细信息。

```
1 du -k 文件夹名：显示的单位为KB
2 du -m 文件夹名：显示单位为MB
3 注：ls -l：可以显示文件大小，不能显示文件夹的大小
```

5.4 频率查询

```
1 目录下 /sys/devices/system/cpu/cpu0/cpufreq/
```

```
sh-4.4# cat /sys/devices/system/cpu/cpu5/cpufreq/cpuinfo_max_freq
1200000
sh-4.4# cat /sys/devices/system/cpu/cpu5/cpufreq/cpuinfo_min_freq
800000
```

5.5 网络查询

网络信息（在网络工具模块）

5.5.1 ifconfig命令

ifconfig -a

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.103  netmask 255.255.255.0  broadcast 192.168.10.255
        inet6 fe80::6571:5b46:c641:41e2  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:53:ff:da  txqueuelen 1000  (Ethernet)
        RX packets 303  bytes 26511 (25.8 KiB)
        RX errors 0  dropped 2  overruns 0  frame 0
        TX packets 4867  bytes 574557 (561.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

网卡

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ens33              网卡名称
UP                 表示“接口已启用”
BROADCAST          表示“主机支持广播”
RUNNING            表示“接口在工作中”
MULTICAST          表示“主机支持多播”
MTU 1500           （最大传输单元）：1500字节
```

ip

```
inet 192.168.xxx.xxx  netmask 255.255.255.0  broadcast 192.168.xxx.xxx
inet:              IP地址
netmask:           子网掩码
broadcast:         广播地址
```

mac

```
ether 00:50:56:28:2c:xx  txqueuelen 1000  (Ethernet)
ether (Ethernet)                    :          表示 连接类型（以太网）
00:50:56:28:2c:xx (Hwaddr)          :          表示 硬件Mac 地址
```

```
4  txqueuelen 1000                  :          表示  网卡传送队列长度
```

包数

RX packets	接受到的总包数
RX bytes	接受到的总字节数
RX errors	接收时，产生错误的数据包数
RX dropped	接收时，丢弃的数据包数
RX overruns	接收时，由于速度过快而丢失的数据包数
RX frame (框架)	接收时，发生frame错误而丢失的数据包数
TX packets	发送的总包数
TX bytes	发送的总字节数
TX errors	发送时，产生错误的数据包数
TX dropped	发送时，丢弃的数据包数
TX overruns	发送时，由于速度过快而丢失的数据包数
TX carrier	发送时， 发生carrier错误而丢失的数据包数（运输工具）
TX collisions	发送时， 冲突信息包的数目

5.5.2 netstat命令

```
1  netstat -natu
```

-a	显示所有选项，默认不显示listen相关
-t	(tcp) 仅显示tcp相关
-u	(udp）仅显示udp相关选项
-n	拒绝显示别名， 能显示数字的全部转化成数字
-l	仅列出有在listen（监听）的服务状态
-p	显示建立相关连接的程序名
-r	显示路由信息， 路由表

-e	显示扩展信息， 例如uid等
-s	按每个协议进行统计
-c	每隔一个固定时间， 执行该netstat命令

5.5.3 route命令

5.5.4 ip命令

ip addr:	显示和管理网络接口的IP地址和相关信息。
ip link:	显示和管理网络接口的状态和属性。
ip route:	显示和管理路由表。
ip neigh:	显示和管理邻居表，即ARP缓存。
ip tunnel:	管理网络隧道。
ip link set:	配置网络接口的状态和属性。
ip addr add/del:	添加或删除IP地址。
ip route add/del:	添加或删除路由。

6. 系统时间

6.1 文件时间

```
pi@raspberrypi:/usrdata/testmonitor $ stat 1
  文件：1
  大小：66              块：8          IO 块：4096   普通文件
设备：b302h/45826d      Inode：1297993     硬链接：1
权限：(0777/-rwxrwxrwx)  Uid：(    0/    root)   Gid：(    0/    root)
最近访问：2022-11-20 22:28:40.830721151 +0800
最近更改：2022-11-23 16:23:36.100283868 +0800
最近改动：2022-11-23 16:23:36.100283868 +0800
创建时间：2022-11-20 22:28:40.830721151 +0800
```

stat file 会出现四个时间，最近问时间，最近更改时间，最近改动时间，创建时间

（1）创建时间，文件创建后创建时间不会改变，ubuntn 18/20不显示创建时间

（2）最近更改时间，touch，echo，cat（树莓派）会更改最近更改时间

（3）最近改动时间，touch，chmod，echo， cat(树莓派)

（4）最近访问时间，touch，cat(树莓派除外)

参与测试的系统ubuntu 18/20, centos 8, edbian11，树莓派

注意是树莓派的 cat与echo功能相同，还不改变最近访问时间

6.2 系统时间

时钟分为系统时钟（System Clock）和硬件（Real Time Clock，简称RTC）时钟。

6.2.1 查看系统时间：

```
1 date -R
```

6.2.2 查看硬件时间:

```
1 sudo hwclock --show
```

6.2.3 修改Ubuntu硬件时间

```
1 前面二个修改系统时间
2 sudo date -s MM/DD/YY       //修改日期
3 sudo date -s hh:mm:ss       //修改时间
4 sudo hwclock --systohc      //修改生效
```

6.2.4 时间函数接口

配置时间：

调用clock_settime、settimeofday函数。

获取时间：

调用time、clock_gettime、gettimeofday函数函数。

6.2.5 启动时间

uptime命令是，

现在时间，启动后运行时间，负载

```
li_jk@ljkpc:~/ljkTest/clibtest$ uptime
 15:18:56 up  3:54,  1 user,  load average: 0.12, 0.12, 0.05
```

cat /proc/uptime

单位s，前面是启动时间，后面是空闲时间

```
li_jk@ljkpc:~/ljkTest/clibtest$ cat /proc/uptime
14071.55 111691.50
```

6.2.6 时区修改

查看时区

```
1 date -R && date +%Z
```

修改时区命令

```
1 tzselect
```

```
1 vim /etc/localtime
```

将UTC改为CST或者UTC+8即可。

或

```
1 rm -f  /etc/localtime
2 ln -s  /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

7. 系统启动

7.1 systemd

创建一个 .service 单元文件，定义服务的配置和启动命令。通常，这些文件存储在 /etc/systemd/system/ 目录下。

使用 systemctl 命令来启用和管理服务。例如，使用以下命令启用服务：

使能：sudo systemctl enable myservice.service　　开始一次就行

启动：sudo systemctl start myservice.service

重启：sudo systemctl restart myservice.service

停止：sudo systemctl stop myservice.service

失能：sudo systemctl disble myservice.service　　删除

查看：sudo systemctl status myservice.service

```
[Unit]
Description=My Service
After=network.target

[Service]
Type=simple
User=root
Group=root
ExecStart=/path/to/my/service

[Install]
WantedBy=default.target
```

system位置

/usr/lib/systemd/system/ 软件包安装的单元

/etc/systemd/system/　　系统管理员安装的单元, 优先级更高

优先级

systemd的使用大幅提高了系统服务的运行效率, 而unit的文件位置一般主要有三个目录

/etc/systemd/system

/run/systemd/system

/lib/systemd/system

这三个目录的配置文件优先级依次从高到低，如果同一选项三个地方都配置了，优先级高的会覆盖优先级低的。

system 命令

```
列出正在运行的 Unit
$ systemctl list-units
列出所有 Unit，包括没有找到配置文件的或者启动失败的
$ systemctl list-units --all
列出所有没有运行的 Unit
$ systemctl list-units --all --state=inactive
列出所有加载失败的 Unit
$ systemctl list-units --failed
列出所有正在运行的、类型为 service 的 Unit
$ systemctl list-units --type=service
显示某个 Unit 是否正在运行
$ systemctl is-active systemd-timesyncd.service

显示某个 Unit 服务是否建立了启动链接
$ systemctl is-enabled systemd-timesyncd.service
立即启动一个服务
$ sudo systemctl start bootlogo.service
立即停止一个服务
$ sudo systemctl stop bootlogo.service
重启一个服务
$ sudo systemctl restart bootlogo.service
杀死一个服务的所有子进程
$ sudo systemctl kill bootlogo.service
重新加载一个服务的配置文件
$ sudo systemctl reload bootlogo.service
重载所有修改过的配置文件
$ sudo systemctl daemon-reload
显示某个 Unit 的所有底层参数
$ systemctl show httpd.service
显示某个 Unit 的指定属性的值
$ systemctl show -p CPUShares avahi-daemon.service
设置某个 Unit 的指定属性
$ sudo systemctl set-property avahi-daemon.service CPUShares=500
命令列出一个 Unit 的所有依赖
$ systemctl list-dependencies avahi-daemon.service
命令列出一个 Unit 的所有依赖，并展开显示 Target 依赖类型
$ systemctl list-dependencies --all avahi-daemon.service

检查某个单元是否是开机自启动的（建立的启动链接）
$ systemctl is-enabled avahi-daemon.service
重启系统（异步操作）
$ systemctl reboot
关闭系统，切断电源（异步操作）
```

```
$ systemctl poweroff
仅 CPU 停止工作，其他硬件仍处于开机状态（异步操作）
$ systemctl halt
暂停系统（异步操作），执行 suspend.target
$ systemctl suspend
使系统进入冬眠状态（异步操作），执行 hibernate.target
$ systemctl hibernate
查看所有日志
$ sudo journalctl
指定日志文件占据的最大空间
$ sudo journalctl --vacuum-size=8M
```

7.2 SysV init

创建一个启动脚本（通常是一个可执行的脚本文件），定义服务的配置和启动命令。这些脚本文件存储在 /etc/init.d/ 目录下。

my-service可以是脚本或者执行文件，必须在这个文件下，不能是软连接

不同定义优先级

0 - 停机（千万不能把initdefault 设置为0 ）

1 - 单用户模式 进入方法#init s = init 1

2 - 多用户，没有 NFS

3 - 完全多用户模式(标准的运行级)

4 - 没有用到

5 - X11 多用户图形模式（xwindow)

6 - 重新启动 （千万不要把initdefault 设置为6 ）

sudo update-rc.d my-service defaults 99

优先级的数字越大，表示越迟运行，这里我们把自己写的服务放在最后运行。

这将在系统启动时自动启动该服务。

7.3 rc.local：

打开 /etc/rc.local 文件

在文件的末尾添加要在启动时运行的命令或脚本。确保命令或脚本的语法正确，并在最后一行添加 exit 0，以表示脚本的结束。

例如，要在启动时运行 /path/to/your/script.sh 脚本，可以在 rc.local 文件中添加以下内容：

```
#!/bin/sh
/path/to/your/script.sh
#/path/to/your/script.sh  &

exit 0
```

```
 #!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

# added by 360Skygo installer
/usr/local/360/canids/can_ids &

exit 0
```

8. 进程线程

8.1 ps命令

```
-A          显示所有进程（等价于-e）(utility)
-a          显示一个终端的所有进程，除了会话引线
-N          忽略选择。
-d          显示所有进程，但省略所有的会话引线(utility)
-x          显示没有控制终端的进程，同时显示各个命令的具体路径。dx不可合用。（utility）
-p          pid 进程使用cpu的时间
-u          uid or username 选择有效的用户id或者是用户名
-g          gid or groupname 显示组的所有进程。
U          username 显示该用户下的所有进程，且显示各个命令的详细路径。如:ps U zhang;
(utility)
-f          全部列出，通常和其他选项联用。如：ps -fa or ps -fx and so on.
-l          长格式（有F,wchan,C 等字段）
-j          作业格式
-o          用户自定义格式。
v          以虚拟存储器格式显示
s          以信号格式显示
-m          显示所有的线程
-H          显示进程的层次(和其它的命令合用，如：ps -Ha)（utility）
e          命令之后显示环境（如：ps -d e; ps -a e）(utility)
h          不显示第一行
```

查看进程

```
ps -ef
ps -e
```

查看线程

```
ps -T -p {}
```

1、ps 输出中，带有方括号 [] 的进程是内核线程

2、前几号进程

PID 1 - init 或 systemd：　　在早期的 Linux 系统中，init 进程是第一个用户空间进程，负责启动和管理其他进程。然而，现代 Linux 发行版通常使用 systemd 作为初始化系统。这个进程是系统启动的根进程，它启动其他系统服务和守护进程。

PID 2 - kthreadd：　　kthreadd 是内核线程，负责创建和管理其他内核线程。它是内核线程的父进程。

PID 3 - kswapd：　　kswapd 是内核线程，用于管理虚拟内存中的页面交换。

PID 4 - md/0：　　md 进程通常与软件 RAID（磁盘冗余阵列）相关，用于管理磁盘阵列。

PID 5 - ksoftirqd/0：　　ksoftirqd 进程用于处理软中断，这是一种在内核中异步执行的任务，通常与网络和输入设备等相关。

PID 6 - kworker/u8:0-events_unbound：　　kworker 进程通常用于处理内核中的异步事件，例如与 CPU 调度相关的任务。

PID 7 - migration/0：　　migration 进程负责处理进程迁移，通常与多核 CPU 相关。

查看进程树

```
1 pstree
```

查看线程个数

```
1 ls /proc/PID/task | wc -l
```

8.2 进程模块

/proc下

cmdline　　进程名

status　　进程号信息

8.3 登陆模块

```
1 /run/var/utmp　　登陆
```

9. 软件安装更新

9.1 apt-get

apt-get update这个命令是用来更新软件列表，会访问软件源列表里的每个网址，并读取软件列表信息，然后保存在本地主机。在软件包管理器里看到的软件列表，都是通过update命令更新的。

- apt-get update：是同步/etc/apt/sources.list和/etc/apt/sources.list.d中列出的软件源的软件包版本，这样才能获取到最新的软件包。
- apt-get upgrade：是更新已安装的所有或者指定软件包，升级之后的版本就是本地索引里的，因此，在执行 upgrade 之前一般要执行update，这样安装的才是最新的版本。

9.2 软件安装

9.2.1 deb安装

在 Debian、 Ubuntu 等 Linux 发行版中，通常使用 deb（debian）形式的软件包

deb**安装**

```
1 sudo dpkg -i xxxx.deb
```

deb**卸载**

先查找软件

```
1 sudo dpkg -l | grep  ***
2 sudo dpkg -l | grep "qq"
```

知道软件名后

```
1 sudo dpkg -r 软件名
2 sudo dpkg -r linuxqq
```

9.2.2 apt安装

apt（Advanced Package Tool）包管理工具，其功能则更加丰富和方便使用，使用 apt 能够

自动从互联网的软件仓库中搜索、安装、升级、卸载软件，它会咨询软件仓库，并能安装软件时

的模块及依赖问题。

```
sudo apt-get install 软件名
```

- apt-get 工具：主要负责软件包的的安装、卸载以及更新等事务。
- apt-cache 工具：用于查询软件包的相关信息。
- apt-config 工具：用于配置所有 apt 工具。

9.2.3 rpm安装

在 RedHat， Fedora， Centos 等派系的 Linux 发行版中，通常使用 rpm（RedHat Package Manager) 形式的软件包，下图是 Vim 软件的 rpm 安装包。

```
rpm -ivh xxxx.rpm
```

9.3 ubuntu建立桌面图标

图标

```
/usr/share/applications
```

下建立配置文件

配置文件示例1

```
[Desktop Entry]
Encoding=UTF-8Version=1.0Type=Application
Name=Electronic WeChat
Icon=electronic-wechat.png
Exec=/opt/electronic-wechat-linux-x64/electronic-wechat
StartupNotify=false
```

```
StartupWMClass=electronic-wechat
OnlyShowIn=Unity;X-UnityGenerated=true12345678
```

配置文件示例2

```
pycharm.desktop
[Desktop Entry]
Type=Application
Name=Pycharm
GenericName=Pycharm3
Comment=Pycharm3:The Python IDE
Exec=sh /home/li_jk/pycharm-community-2021.3.3/bin/pycharm.sh
Icon=/home/li_jk/pycharm-community-2021.3.3/bin/pycharm.png
Terminal=pycharm
Categories=Pycharm;
```

10. 开源协议

10.1 许可证说明

GNU通用公共许可证（GNU General Public License，GPL）

GPL是最常见和最具影响力的开源协议之一。它有多个版本，包括GPLv2和GPLv3。它要求任何基于或修改自GPL许可的软件的派生作品都必须以相同的协议分发。GPL保证了用户可以自由地使用、修改和分发软件，同时确保了这些自由权利在未来也不会被剥夺。

MIT许可证

MIT许可证是一种非常宽松的开源协议。它允许自由地使用、修改和分发软件，无论是作为源代码还是作为编译后的二进制代码。MIT许可证对派生作品的许可要求相对较少。允许您在商业项目中使用、修改和分发开源代码，无论是作为源代码还是二进制形式。与GPL不同，MIT许可证没有要求派生作品必须使用相同的许可证进行分发。

Apache许可证

Apache许可证也是一种宽松的开源协议。它允许自由地使用、修改和分发软件，并包括对专利权的明确授权。Apache许可证对派生作品的许可要求相对较少。允许您在商业项目中自由使用、修改和分发开源代码。它还提供了对专利权的明确授权。

BSD许可证

BSD许可证是一系列类似的许可证，如BSD 2-Clause License和BSD 3-Clause License。BSD许可证也是一种宽松的许可证，允许自由地使用、修改和分发软件。与MIT和Apache许可证类似，BSD许可证对派生作品的许可要求相对较少。这些许可证允许您在商业项目中使用、修改和分发开源代码，而且对派生作品的许可要求相对较少。

MPL Mozilla公共许可证（Mozilla Public License，MPL）

MPL是一种相对较为复杂的开源协议，适用于涉及Mozilla项目的软件。MPL要求对源代码的修改必须以MPL或兼容协议进行分发，但对于以MPL许可的原始代码的直接使用并没有强制要求。

10.2 主流开源协议的一些典型开源项目

GNU通用公共许可证（GPL）

Linux内核：世界上最著名的开源项目，操作系统内核。

GCC（GNU Compiler Collection）：一套广泛使用的编译器集合。

GNU工具链：包括GNU Binutils、GDB调试器等工具。

GIMP（GNU Image Manipulation Program）：图像编辑和处理软件。

LibreOffice：办公套件，包括文档处理、电子表格、演示文稿等功能。

MIT许可证

Node.js：基于Chrome V8引擎构建的JavaScript运行时环境。

Ruby on Rails：基于Ruby语言的Web应用程序框架。

jQuery：JavaScript库，简化了客户端脚本编写。

React Native：用于构建跨平台移动应用的JavaScript框架。

Xamarin.Forms：用于创建跨平台移动应用的.NET开发工具。

Apache许可证

Apache HTTP服务器：世界上最流行的Web服务器软件。

Hadoop：用于分布式存储和处理大规模数据集的开源框架。

Tomcat：Java Servlet和JavaServer Pages (JSP)容器。

Cassandra：分布式数据库系统，用于处理大规模数据集。

Kafka：高性能、可扩展的分布式消息队列系统。

BSD许可证

FreeBSD：基于BSD操作系统的自由和开放源代码的操作系统。

NetBSD：可移植的操作系统，具有高度可扩展性。

OpenBSD：注重安全性的自由和开放源代码的操作系统。

PostgreSQL：关系型数据库管理系统。

Nginx：高性能的Web服务器和反向代理服务器。

Mozilla公共许可证（MPL）

Mozilla Firefox：自由和开放源代码的Web浏览器。

Thunderbird：自由和开放源代码的电子邮件和新闻客户端。

Rust：系统级编程语言，注重安全性和并发性。

VLC媒体播放器：跨平台的多媒体播放器和流媒体服务器。

GUN、MIT、Apache、BSD、MPL各种开源协议介绍和区别_mpl协议-CSDN博客

11. ARM架构

11.1 架构

11.1.1 ARM版本Ⅰ： V1版架构

该版架构只在原型机ARM1出现过，只有26位的寻址空间，没有用于商业产品。

其基本性能有:

☒基本的数据处理指令（无乘法）；

☒基于字节、半字和字的Load/Store指令;

☒转移指令，包括子程序调用及链接指令；

☒供操作系统使用的软件中断指令SWI;

☒寻址空间：64MB（226）。

11.1.2 ARM版本 II： V2版架构

该版架构对V1版进行了扩展，例如ARM2和ARM3（V2a）架构。包含了对32位乘法指令和协处理器指令的支持。 版本2a是版本2的变种，ARM3芯片采用了版本2a，是第一片采用片上Cache的ARM处理器。同样为26位寻址空间，现在已经废弃不再使用。

V2版架构与版本V1相比，增加了以下功能：

☒乘法和乘加指令；

☒支持协处理器操作指令；

☒快速中断模式；

☒SWP/SWPB的最基本存储器与寄存器交换指令；

☒寻址空间：64MB。

11.1.3 ARM版本III： V3版架构

ARM作为独立的公司，在1990年设计的第一个微处理器采用的是版本3的ARM6。它作为IP核、独立的处理器、具有片上高速缓存、MMU和写缓冲的集成CPU。 变种版本有3G和3M。版本3G是不与版本2a向前兼容的版本3，版本3M引入了有符号和无符号数乘法和乘加指令，这些指令产生全部64位结果。V3版架构（目前已废弃）对ARM体系结构作了较大的改动：

☒寻址空间增至32位（4GB）；

☒当前程序状态信息从原来的R15寄存器移到当前程序状态寄存器CPSR中（Current Program Status Register）；

☒增加了程序状态保存寄存器SPSR（Saved Program Status Register）；

☒增加了两种异常模式，使操作系统代码可方便地使用数据访问中止异常、指令预- - 取中止异常和未定义指令异常。；

☒增加了MRS/MSR指令，以访问新增的CPSR/SPSR寄存器；

☒增加了从异常处理返回的指令功能。

11.1.4 ARM版本IV： V4版架构

V4版架构在V3版上作了进一步扩充，V4版架构是目前应用最广的ARM体系结构，ARM7、ARM8、ARM9和StrongARM都采用该架构。 V4不再强制要求与26位地址空间兼容，而且还明确了哪些指令会引起未定义指令异常。

指令集中增加了以下功能：

☒符号化和非符号化半字及符号化字节的存/取指令；

⊠增加了T变种，处理器可工作在Thumb状态，增加了16位Thumb指令集；

⊠完善了软件中断SWI指令的功能；

⊠处理器系统模式引进特权方式时使用用户寄存器操作;

⊠把一些未使用的指令空间捕获为未定义指令

11.1.5 ARM版本Ⅴ： V5版架构

V5版架构是在V4版基础上增加了一些新的指令，ARM10和Xscale都采用该版架构。

这些新增命令有：

⊠带有链接和交换的转移BLX指令；

⊠计数前导零CLZ指令； BRK中断指令；

⊠增加了数字信号处理指令（V5TE版）；

⊠为协处理器增加更多可选择的指令；

⊠改进了ARM/Thumb状态之间的切换效率；

⊠E—增强型DSP指令集，包括全部算法操作和16位乘法操作；

⊠J----支持新的JAVA，提供字节代码执行的硬件和优化软件加速功能。

11.1.6 ARM版本Ⅵ： V6版架构

V6版架构是2001年发布的，首先在2002年春季发布的ARM11处理器中使用。在降低耗电量地同时，还强化了图形处理性能。通过追加有效进行多媒体处理的SIMD(Single Instruction, Multiple Data，单指令多数据)功能，将语音及图像的处理功能提高到了原型机的4倍。

此架构在V5版基础上增加了以下功能：

⊠THUMBTM：35%代码压缩；

⊠DSP扩充：高性能定点DSP功能；

⊠JazelleTM：Java性能优化，可提高8倍；

⊠Media扩充：音/视频性能优化，可提高4倍

11.1.7 ARM版本Ⅶ： V7版架构

V7架构是在ARMv6架构的基础上诞生的。该架构采用了Thumb-2技术,它是在ARM的Thumb代码压缩技术的基础上发展起来的, 并且保持了对现存ARM解决方案的完整的代码兼容性。Thumb-2技术比纯32位代码少使用31%的内存,减小了系统开销。同时能够提供比已有的基于Thumb技术的解决方案高出38%的性能。

ARMv7架构还采用了NEON技术,将DSP和媒体处理能力提高了近4倍,并支持改良的浮点运算,满足下一代3D图形、游戏物理应用以及传统嵌入式控制应用的需求。此外,ARMv7还支持改良的运行环境,以迎合不断增加的JIT(Just In Time)和DAC(DynamicAdaptive Compilation)技术的使用。

11.1.8 ARM版本ⅤⅢ： V8版架构

v8架构是在32位ARM架构上进行开发的，将被首先用于对扩展虚拟地址和64位数据处理技术有更高要求的产品领域，如企业应用、高档消费电子产品。ARMv8架构包含两个执行状态：AArch64和AArch32。AArch64执行状态针对64位处理技术，引入了一个全新指令集A64；而AArch32执行状态将支持现有的ARM指令集。目前的ARMv7架构的主要特性都将在ARMv8架构中得以保留或进一步拓展，如：TrustZone技术、虚拟化技术及NEON advanced SIMD技术等。

架构	处理器家族
ARMv1	ARM1
ARMv2	ARM2、ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM、ARM7TDMI、ARM9TDMI
ARMv5	ARM7EJ、ARM9E、ARM10E、XScale
ARMv6	ARM11、ARM Cortex-M
ARMv7	ARM Cortex-A、ARM Cortex-M、ARM Cortex-R
ARMv8	Cortex-A50[9]

11.2 指令

12. 常见问答

南向、北向生态库

交叉工具链是用于在一种处理器架构（通常为主机机器）上生成另一种处理器架构（目标机器）上的可执行文件的工具链。

在北向和南向生态中，北向生态指的是基于操作系统的上层应用程序，这些应用程序通常运行在主机机器上，例如运行在Windows或Linux等操作系统上的应用程序。而南向生态指的是运行在底层设备上的软件和固件，例如运行在嵌入式设备上的软件和驱动程序等。

因此，在交叉工具链中，北向生态库是指用于在主机机器上编写和运行的库和头文件，这些库和头文件是用于构建目标机器上的应用程序的。而南向生态库则是指用于在目标机器上编写和运行的库和头文件，这些库和头文件是用于构建运行在底层设备上的软件和固件的。

gdb为什么可以调试程序

GDB（GNU Debugger）是一个功能强大的调试器，用于在程序运行时跟踪和调试代码。它可以帮助开发人员找到和修复程序中的错误和问题。GDB之所以能够调试程序，是因为它利用了操作系统和编译器提供的一些特性和机制。

原理如下：

符号表：在编译程序时，编译器会生成一个符号表。符号表包含了程序中所有变量、函数和代码地址的映射关系。GDB通过读取符号表，可以获得源代码与机器代码之间的对应关系，从而能够在调试过程中将机器代码映射回源代码。

调试信息：编译器在生成可执行文件时，会将调试信息（Debug Information）嵌入到可执行文件中。调试信息包含了源代码的行号信息、变量的类型和名称等。GDB通过读取调试信息，可以获得与源代码和变量相关的信息，从而能够在调试过程中显示源代码的行号、查看变量的值等。

调试器接口：操作系统提供了一些调试器接口，使得GDB能够与运行中的程序进行交互。通过这些接口，GDB可以控制程序的运行、暂停程序的执行、查看和修改内存中的数据等。

断点：GDB可以在程序中设置断点，即在特定的代码位置暂停程序的执行。当程序运行到断点处时，GDB会自动暂停程序的执行，使得开发人员可以检查程序状态和变量值。

在Linux系统下，GDB使用ptrace系统调用来与被调试程序进行通信。ptrace允许一个进程（调试器）监控另一个进程（被调试程序）的执行，并可以读取和修改被调试进程的寄存器、内存以及程序状态。

软中断

软中断（Software Interrupt）是由程序中的特定指令或软件调用触发的中断，用于请求操作系统内核提供某种服务或执行特定的功能。软中断是与硬件中断相对应的概念，在执行软中断时，CPU会切换到内核态并执行相应的中断处理程序，然后再返回用户态继续执行程序。

在x86架构的CPU上，软中断使用int指令来触发，通过将中断向量号传递给int指令来指示触发哪个软中断。常见的软中断包括：

0x80（INT 0x80）：	在Linux操作系统中，用于执行系统调用，请求内核提供各种服务，如文件操作、进程管理、网络通信等。
	在某些特定的操作系统中，可能会用于自定义的软中断。

0x81（INT 0x81）：	
0x82（INT 0x82）：	类似于0x81，也是用于自定义的软中断。
0x3（INT 0x3）：	通常称为断点软中断，用于调试程序，在调试器中设置断点时会触发该软中断。

中断上下部分

中断（Interrupt）是计算机系统中的一种事件处理机制，用于在某个时间点暂停正在执行的程序，转而执行处理该事件的中断服务程序（Interrupt Service Routine，简称ISR）。中断分为上半部（Top Half）和下半部（Bottom Half），这两部分构成了中断处理的完整过程。

上半部（Top Half）： 上半部是中断处理的第一阶段，它负责执行紧急且必要的中断处理操作，通常是在最短时间内完成。上半部的任务包括：

保存CPU当前的上下文：将当前执行的程序的状态（寄存器、程序计数器等）保存起来，以便稍后恢复。

执行中断处理程序：跳转到预先定义好的中断服务程序（ISR），该程序用于处理中断引发的事件。

尽可能快速地完成处理：上半部要尽可能快速地完成中断处理，以便尽早恢复被中断的程序执行。

下半部（Bottom Half）： 下半部是中断处理的第二阶段，它负责执行较为复杂和耗时的中断处理操作，通常不能在中断上下文中执行。因为中断上下文需要尽可能地快速执行，所以某些处理可能无法在上半部完成，这时需要将这些处理推迟到下半部执行。下半部的任务包括：

延迟处理：将一些复杂或需要长时间执行的中断处理操作延迟到下半部执行。

唤醒进程：如果中断事件导致某些进程被阻塞，下半部可以唤醒这些进程继续执行。

执行延迟工作：处理一些需要延迟执行的任务，例如网络数据包接收、磁盘IO等。

在Linux内核中，上半部通常在中断上下文中执行，而下半部则在软中断或任务队列

irq是什么中断

IRQ（Interrupt Request）是一种计算机系统中的硬件中断，用于通知CPU有外部设备请求处理或发生了特定事件。当外部设备或事件需要处理时，它会向CPU发送一个中断请求信号，CPU在接收到IRQ信号后会立即中断当前正在执行的任务，转而执行相应的中断处理程序（ISR）来处理该事件。

IRQ中断是一种异步的事件触发机制，它不依赖于CPU的主动轮询，而是在外部设备发生请求或事件发生时被动地触发。每个IRQ中断都有一个唯一的中断号（IRQ number），用于标识不同的中断源或外

部设备。在计算机系统中，IRQ中断通常被用于处理各种外部设备的请求，例如硬件设备（如网卡、磁盘控制器、键盘、鼠标等）的中断请求，以及其他一些特定事件的通知（如时钟中断、系统调用等）。

当IRQ中断发生时，CPU会执行以下步骤：

保存当前执行任务的上下文：CPU会将当前正在执行的程序的寄存器和程序计数器等状态保存到堆栈中，以便稍后恢复执行。

根据中断号查找ISR：CPU根据接收到的IRQ中断号，在中断向量表（Interrupt Vector Table）中查找相应的中断服务程序（ISR）的地址。

执行ISR：CPU跳转到找到的ISR地址，开始执行中断服务程序来处理IRQ中断请求。

中断处理完成后恢复现场：ISR执行完成后，CPU会从堆栈中恢复之前保存的状态，回到被中断的任务继续执行。