block:磁盘上存放的数据块
block bitmap:在元数据区域,标记数据块是否被使用
文件系统:软件(XFS NTFS FAT EXT3 EXT4等),存储数据或文件的一种格式,文件系统不存在分区上,位于磁盘的某一个位置;文件系统把分区分割为两部分,一部分存放元数据(metadata),另一部分存放数据
metadata:元数据(和数据本身内容没有关系,包括的是数据属性,比如数据归属,数据权限,数据的时间戳)
inode:index node(索引节点),存放数据的属性,不包括文件名,数据对应的数据块(block),同一个inode可以指向多个文件
super block:超级块,管理元数据区域内其他的区域的
ln(link):硬链接和软链接

ln [-s -v]:语法: ln [-s -v] src_file des_file
 -s:创建软链接
 -v:显示详细过程

硬链接:只能对文件进行硬链接,并且在同一个文件系统,指向一个inode节点
软链接:使用绝对路径,指向一个路径,文件大小是字符串的个数,跨文件系统建立,可以为文件或目录建立

删除文件:把数据对应的inode和block的值变为0,实际数据并没用删除
新建文件:
复制文件:重新建立新文件
移动文件:改变inode和block值,对数据没有操作

图书馆找书: 1万册藏书

目录:书:编号:书名 位置

001:Linux 第三排位置

100万册藏书: 书分类:计算机:Linux Windows Unix等 工程 科学

10M:10份, 1M

inode:10000 block:100 101 102 a.txt

touch: b.txt

文件开端:根 / 一切皆文件 根文件系统自检测

/backup/abc/gooann.txt

/opt/gooann.txt

目录也是文件

/root/a.txt--->/opt/a.txt

链接文件:

设备文件: /dev/
  b:块设备
  c:字符设备

  主设备号(MAJOR):标识某一类设备
  次设备号(MINOR):标识一类设备下不同设备

mknod:创建设备:mknod [OPTION]... NAME TYPE [MAJOR MINOR]
 -m：设置权限

磁盘:
IDE ATA:hd 主板:2IDE接口
SATA:sd
SCSI:sd
USB:sd

Linux以字母标识磁盘的个数 a:第一块 b:第二块

Linux用数字标识分区:1-4标识主分区或扩展分区 逻辑分区从5开始

sda1

低级格式化:划分磁道
高级格式化:为分区装载文件系统
文件系统:内核功能

FAT32 NTFS EXT2 EXT3 EXT4 XFS等

mkfs:make file system创建文件系统
 -t:指定文件系统
 mkfs -t ext3 dev_name

vfs:虚拟文件系统

fdisk:针对MBR分区类型的工具
gdisk:针对GPT分区类型的工具

fdisk:交互式工具
  d:删除一个分区
  l:显示linux支持的分区类型
  m:显示帮助信息
  n:创建一个新分区
    p:创建主分区
      e:创建扩展分区
  p:显示当前分区列表
  q:不保存退出
  w:保存更改并退出
  t:更改分区的系统ID

Centos6:按照柱面分区
Centos7:按照扇区分区

磁盘ID:
 5:扩展分区
 82:交换分区
 83:linux系统分区
 8e:LVM(逻辑卷管理)

/proc/partitions:查看系统分区情况

mkfs -t xfs /dev/sdb1
mkfs.xfs /dev/sdb1

/
/usr
/etc/
/var
/bin
/tmp

/var/gooann.txt

/var/abc/gooann

sdb1 abc/gooann

mount:挂载工具
  mount src des

/sdb1 mount /dev/sdb1 /sdb1

实现开机自动挂载:
/etc/fstab添加内容:
UUID=id号 挂载点 文件系统 默认选项 是否开机检查 是否备份
UUID=8567c02b-fac2-46d3-984e-b9fd8c9a2072 /sdb1 xfs defaults 0 0

mkswap:创建交换分区

blkid:查看设备UUID

软件包管理:
软件包管理工具:RPM

应用程序:和CPU架构密不可分的

源代码--编译--链接--运行

链接:
库:静态库和动态库 .so shared object
静态链接过程:把库文件集成到应用程序中
动态链接过程:库文件和应用程序分开的,做链接

编译:生成二进制文件(可执行文件),匹配底层架构

程序=数据+指令 应用程序运行时,申请CPU和内存资源

内存地址空间:
text:存放指令
data+BSS:data存放数据,BSS:block system of symbol:存放初始化为0的变量
堆:heap存放运行中临时文件
栈:stack存放应用程序的变量和函数等

应用程序中:
二进制文件(可执行程序)
库
配置文件
帮助文件

/bin

/sbin
/etc/

/usr/bin
/usr/sbin

/usr/local:独立

软件包依赖关系:

A---B---C:安装软件先安装被依赖软件包,卸载软件包时,先卸载依赖的软件

软件包管理器的核心功能:
1、制作软件包;
2、安装、卸载、升级、查询、校验;

Redhat, SUSE, Debian

Redhat, SUSE: RPM
    Redhat Package Manager
    PRM is Package Manager
Debian: dpt

前端工具：yum, apt-get,自行解决依赖关系
后端工具：RPM, dpt

RPM优点:
1:RPM软件包中包含了编译过的程序与配置文件,用户不用重新编译
2:RPM在安装之前,首先会检查硬盘容量,操作系统版本等,避免被错误安装
3:RPM使用RPM数据库管理,数据中记录应用程序的参数,便于升级,卸载,查询等,RPM数据库(/var/lib/rpm)
4:RPM本身会提供应用程序版本信息,相关属性,软件名称,用途等,便于用户了解软件

RPM功能:安装、查询、卸载、升级、校验、数据库的重建、验正数据包等工作;

rpm命名:
包：组成部分
    主包:
        bind-9.7.1-1.el5.i586.rpm
    子包:
        bind-libs-9.7.1-1.el5.i586.rpm

bind-utils-9.7.1-1.el5.i586.rpm

包名格式：

name-version-release.arch.rpm

bind-major.minor.release-release.arch.rpm

主版本号:功能重大改进修改主版本号

次版本号:某个子功能发生重大变化

发行号:修正了部分bug,调整了一点功能

RPM包:经过源代码编译后的软件包

.tar.gz:源码包,没有经过编译

1:安装:rpm -ivh name.rpm

-i:安装软件包

-h:以#显示进度；每个#表示2%;

-v:显示详细过程

rpm -ivh /PATH/TO/PACKAGE_FILE

--nodeps:忽略依赖关系,可能导致软件包安装不完整

--replacepkgs:重新安装,替换原有安装；

--force:强行安装,可以实现重装或降级；

2、查询

rpm -q PACKAGE_NAME： 查询指定的包是否已经安装

rpm -qa : 查询已经安装的所有包

rpm -qi PACKAGE_NAME: 查询指定包的说明信息；

rpm -ql PACKAGE_NAME: 查询指定包安装后生成的文件列表；

rpm -qc PACEAGE_NEME：查询指定包安装的配置文件；

rpm -qd PACKAGE_NAME: 查询指定包安装的帮助文件；

rpm -qf /path/to/somefile: 查询指定的文件是由哪个rpm包安装生成的；

rpm -q --scripts PACKAGE_NAME: 查询指定包中包含的脚本

脚本:包含软件包安装前(preinstall),安装后(postinstall),卸载前(preuninstall),卸载后(postuninstall)执行的脚本

如果某rpm包尚未安装，我们需查询其说明信息、安装以后会生成的文件；

rpm -qpi /PATH/TO/PACKAGE_FILE:查看未安装软件包信息

rpm -qpl :查看未安装软件包列表

3、升级

rpm -Uvh /PATH/TO/NEW_PACKAGE_FILE: 如果装有老版本的，则升级；否则，则安装；

rpm -Fvh /PATH/TO/NEW_PACKAGE_FILE：如果装有老版本的，则升级；否则，退出；

--oldpackage: 降级

4、卸载
rpm -e PACKAGE_NAME
--nodeps:卸载时忽略依赖关系

5、校验:验证软件包列表或配置文件的完整性
rpm -V PACKAGE_NAME
应用程序配置文件完整性破坏以后的代码:
S file Size differs
M Mode differs (includes permissions and file type)
5 digest (formerly MD5 sum) differs
D Device major/minor number mismatch
L readLink(2) path mismatch
U User ownership differs
G Group ownership differs
T mTime differs
P caPabilities differ

6、重建数据库
rpm
--rebuilddb: 重建数据库，一定会重新建立；
--initdb:初始化数据库，没有才建立，有就不用建立；

7、检验来源合法性，及软件包完整性；

ls /etc/pki/rpm-gpg/
RPM-GPG-KEY-redhat-release

rpm -K /PAPT/TO/PACKAGE_FILE
dsa, gpg: 验正来源合法性，也即验正签名；可以使用--nosignature，略过此项
sha1, md5: 验正软件包完整性；可以使用--nodigest，略过此项

rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release: 导入密钥文件

RPM不能解决软件依赖关系
YUM:Yellowdog Updater Modified

yum解决依赖关系,RPM进行安装卸载等操作

yum采用C/S架构,依靠yum仓库

ftp
web
file

RPM包:功能 依赖关系

yum配置文件:/etc/yum.conf

yum仓库:/etc/yum.repos.d

yum:功能
install:安装软件包
update:升级软件包
check-update:检查软件包的升级信息
upgrade:更新软件包
remove:卸载软件包
list:显示所有已经安装和未安装的软件包
info:查看软件包信息
clean:清除软件包一些信息
search:查看指定软件包相关的软件包
deplist:查询软件包依赖关系列表
repolist:查看可用的yum源
repoinfo:查看可用yum源的信息
groupinstall:安装软件包组
groupinfo:查看软件包组的信息
grouplist:显示所有已经安装和未安装的软件包组
groupremove:卸载指定软件包组

yum源:
[name]
name=: //引用变量
baseurl: //yum仓库
enabled: //是否启用该yum仓库
gpgcheck: //是否进行完整性和校验和检测
gpgkey: //完整性和检验和检测文件

yum元数据目录:repodata
primary.xml.gz:所有RPM文件列表;依赖关系;软件包安装列表
filelists.xml.gz:包含所有RPM包的所有列表
other.xml.gz:包含软件包其他信息,比如更改记录
repomd.xml:包含primary/filelist/other时间戳和检验和
comps.xml:包含软件包组的列表

创建yum元数据目录:

yum -y install createrepo
createrepo /opt/repo

yum仓库:
$releasever:发行版本
$basearch:cpu架构集
$arch:cpu架构

国内yum源列表如下：

1. 企业贡献：
搜狐开源镜像站：http://mirrors.sohu.com/
网易开源镜像站：http://mirrors.163.com/

2. 大学教学：
北京理工大学：
http://mirror.bit.edu.cn (IPv4 only)
http://mirror.bit6.edu.cn (IPv6 only)
北京交通大学：
http://mirror.bjtu.edu.cn (IPv4 only)
http://mirror6.bjtu.edu.cn (IPv6 only)
http://debian.bjtu.edu.cn (IPv4+IPv6)
兰州大学：http://mirror.lzu.edu.cn/
厦门大学：http://mirrors.xmu.edu.cn/
清华大学：
http://mirrors.tuna.tsinghua.edu.cn/ (IPv4+IPv6)
http://mirrors.6.tuna.tsinghua.edu.cn/ (IPv6 only)
http://mirrors.4.tuna.tsinghua.edu.cn/ (IPv4 only)
天津大学：http://mirror.tju.edu.cn/
中国科学技术大学：
http://mirrors.ustc.edu.cn/ (IPv4+IPv6)
http://mirrors4.ustc.edu.cn/
http://mirrors6.ustc.edu.cn/
东北大学：
http://mirror.neu.edu.cn/ (IPv4 only)
http://mirror.neu6.edu.cn/ (IPv6 only)
电子科技大学：http://ubuntu.uestc.edu.cn/

192.168.10.10:yum源
ftp: # mount /dev/cdrom /media
yum -y install vsftpd
systemctl start vsftpd

```
        # systemctl enable vsftpd
        # cp -rf /media/ /var/ftp

httpd:
    # mount /dev/cdrom /media
        # yum -y install httpd
        # systemctl start httpd
        # systemctl enable httpd
        # cp -rf /media/ /var/www/html/

192.168.10.20:yum客户端
ftp:
  vim ftp.repo
  [ftp]
  name=ftp_repo
  baseurl=ftp://192.168.10.10/media
  enabled=1
  gpgcheck=1
  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

httpd:
  # vim httpd.repo
  [httpd]
  name=httpd_repo
  baseurl=http://192.168.10.10/media
  enabled=1
  gpgcheck=1
  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

file:
  # mount /dev/cdrom /media
  # vim file.repo
  [file]
  name=file_repo
  baseurl=file:///media
  enabled=1
  gpgcheck=1
  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

EPEL:扩展yum源
  # yum -y install epel-release
```

ftp:Linux默认共享目录/var/ftp
httpd:Linix默认访问目录/var/www/html/

编译安装: RPM YUM

RPM:一键安装,方便,二进制程序
 1:功能不齐全,功能冗余
 2:版本滞后

 apache:源代码:2.4.1 RPM:2.2.6

程序:源代码--编译--链接--运行

定制软件包:源代码编译安装应用程序

编译环境:
gcc:GNU C Complier
gcc-c++:

安装编译环境: # yum -y groupinstall "Development Tools"

make:不是编译器,项目管理工具,按照Makefile文件中的定义,去定义软件中功能先后顺序等

automake:Makefile.in(半成品)---Makefile

autoconf:生成脚本文件,configure

执行编译安装步骤:
1:获取源代码并解压:
  # tar xf xxx.tar.gz
  # tar xf xxx.tar.bz2
2:执行configure脚本(在源码目录中运行)
  功能:1:检查应用程序所需要的编译环境 2:生成Makefile 3:定制应用程序功能
  选项:--help:查看该脚本帮助信息
  --prefix=PATH:定义应用程序安装路径
  --sbin-path=PATH:定义应用程序的二进制程序路径
  --conf-path=PATH:定义应用程序配置文件路径
  --enable--XXXX:开启某项功能(如应用程序支持该功能,但默认不安装)
  --disable--XXX:关闭某项功能(如应用程序支持该功能,默认安装,但是用户不需要)
  --with--XXX:开启某项功能(如应用程序支持该功能,但默认不安装)
  --without--XXX:关闭某项功能(如应用程序支持该功能,默认安装,但是用户不需要)
  --user=USER:指定应用程序执行的用户
  --group=GROUP:执行应用程序执行的组

解决configure错误信息
  1:用without关闭这项功能
  2:安装所需的环境,安装所需环境的devel包

3:make:定义Makefile按照哪种书序去编译源程序
  -j N:多个线程一起编译

4:make install:安装应用程序

编译安装完成后工作:
1:添加应用程序的二进制程序到系统路径: 系统查找 $PATH
  1.1:为二进制程序做链接文件
    # ln -s /usr/local/tengine/sbin/* /usr/sbin
    # ln -s /usr/local/tengine/sbin /usr/sbin/tengine
  1.2:修改/etc/profile文件
  1.3:在/etc/profile.d/目录中建立name.sh的文件
2:添加应用程序的库文件到系统库文件路径: 库文件:/usr/lib /usr/lib64
  2.1:为库文件做链接文件
    # ln -s /usr/local/APP_NAME/lib/* /usr/lib
    # ln -s /usr/local/APP_NAME/lib /usr/lib/APP_NAME
  2.2:在/etc/ld.so.conf.d/目录下建立APP_NAME.conf,把应用程序的库文件位置添加进去
    # ldconfig
      -v:显示加载库文件过程
3:添加用用程序的头文件到系统头文件路径:系统头文件: /usr/include
  3.1:为头文件做链接文件
    # ln -s /usr/local/APP_NAME/include/* /usr/include
    # ln -s /usr/local/APP_NAME/include /usr/include/APP_NAME
4:添加应用程序的帮助信息:
  4.1:man -M PATH bin_name:-M指定帮助信息的位置
  4.2:man配置文件:系统默认帮助/usr/share/man
      CentOS6:/etc/man.conf添加:MANPATH [help_path]
      CentOS7:/etc/man_db.conf添加:MANPATH_MAP [help_path]

进程管理:
进程:应用程序或者数据在系统中按照顺序执行的活动过程,是操作系统架构基础;磁盘中应用程序或者数据在内存中的映射
OS:kernel+process:进程只能运行在CPU和内存中

进程特性:动态性 并发性等

kernel:内存管理 网络管理 安全功能 驱动程序 进程管理等

CPU分环运行:
0环:特权环,运行内核(内核空间)
1-2环:库文件
3环:进程环(用户空间)

CPU在一个时间点只能运行一个应用程序,CPU按照时间片为应用程序分配运行时间
mkdir: mkdir /data: system call:系统 I/O

CPU:暂时的退出

内存:用户空间和内核空间

task struct:进程的数据结构(存放在内核中)
task struct结构:状态 进程信息和内核栈 运行列表(指令等) MM(内存管理):memory mangement PID:进程ID号 群组信息 用户信息 文件系统 文件描述符等

线性地址

物理地址

MMU:memory mangement unit:内存管理单元

TLB

context switch:进程切换就是上线文切换

进程内存结构:forbidden:内核空间
program text:指令
data+bss:全局变量
heap:堆空间,进程运行时调用的数据等
shared library:共享库
stack:栈空间,局部变量

thread:线程,轻量级进程

运行进程的模式:多进程模式(每一个请求生成一个进程:进程切换)
单进程多线程:每个线程去响应一个请求,线程所使用的资源是进程的资源

lock:死锁 自旋锁等

task_uninterruptible:不可中断睡眠 mkdir:系统调用,硬件:I/O 1ms
task_interruptible:可中断睡眠
stopped:停止状态(不能被调入CPU运行)
zombie:僵尸状态(结束父进程才能够退出当前进程的僵尸状态):进程运行完成后没有释放CPU和内存资源

进程:父子关系 进程状态 优先级

进程优先级:0-139数字,共140个优先级,数字越小,优先级越高
0-99:系统分配优先级(用户不可控优先级)
100-139:nice值 (用户可控优先级);-20到19;root用户可以任意调整nice值;普通用户只能调整0-19
高优先级进程:获取更多的CPU运行时长,更优先的让CPU执行

大O标准: O(1) O(n) O(logn) O(2^n) O(n^2)

进程分类:
和终端相关的进程:用户进程
和终端无关的进程:内核进程

ps:process status:进程状态查看,运行ps那个时刻系统进程状态;BSD风格(不使用连字符),sys V风格(使用连字符),GNU长选项(使用两个连字符)
a:显示和终端有关的进程
u:显示进程用户的信息
x:显示和终端无关的进程
-e:显示所有进程信息
-l:显示进程长格式
-F:显示进程的完整信息

ps aux:结果解析
USER:该进程是有哪个用户发起的
PPID:父进程号
PID:进程号 /proc目录存放着进程相关的信息 在CentOS7:PID为1的进程是systemd 在CentOS6中PID为1的进程是init进程,PID为1的进程是系统中所有进程的父进程
%CPU:该进程占用的CPU百分比
%MEM:该进程占用内存的百分比
VSZ:虚拟内存集(计算进程所占物理内存时,物理内存包括共享库的空间)
RSS:常驻内存集(进程中不能被交换出去的数据)
TTY:运行在哪个终端
STAT:进程状态
进程状态:
D:不可中断睡眠状态
S:可中断睡眠状态
R:运行状态
T:停止状态
Z:僵尸状态
s:session leader:会话的领导者
<:优先级较高的进程(用户不可控)
N:优先级较低的进程(用户可控进程)

l:多线程进程
L:在内存中锁定分页
+:前台进程组中的进程
START:进程启动时间
TIME:该进程在CPU中实际运行的时长
COMMAND:该进程命令名称

ps -elF:结果解析
PRI:系统分配的优先级(用户不可更改)
NI:nice值
调整进程的nice值:
调整正在运行进程的nice
renice [nice_num] PID
指定进程运行的nice值(让某个命令以指定nice运行)
nice -n [nice_num] COMMAND

top:动态查看进程状态,每隔3秒刷新一次进程状态信息
top结果解析:
第一行:当前系统时间 系统运行时长 登录系统的用户个数 过去1分钟,5分钟,15分钟平均负载
第二行:进程总数 正在运行的进程数 睡眠的进程数 停止状态进程数 僵尸状态进程数
第三行(按1键可以查看每个CPU的信息):CPU相关信息: us:用户进程所占CPU百分比 sy:系统进程所占CPU百分比 ni:nice所占CPU百分比 id:空闲进程所占CPU百分比 hi:硬件中断所占CPU百分比 hi:软件中断所占CPU百分比 st:被hypervisor偷走的CPU的百分比
第四行:物理内存相关信息
第五行:交换分区相关信息
top交互式命令:
M:按所占内存百分比进行排序
P:按所占CPU备份比进行排序
T:按进程运行时长进行排序
c:是否显示COMMAND列完整路径
l:是否显示top第一行
t:是否显示top中第二行和第三行
m:是否显示内存的相关信息
k:杀死某个进程
q:退出top进程

pstree:查看进程树(查看父子进程关系)
pgrep:查看某个进程PID号 pgrep process_name
pidof:查看某个进程PID号 pidof process_name
kill:杀死某个进程(向进程传递某个信号)
kill -l:查看信号列表
信号列表:
1:SIGHUP:不停止服务的情况下,重新读取配置文件,并且应用配置文件中的设置
2:SIGINT:Ctrl+c 中断某个进程

9:SIGKILL:强制杀死某个进程
15:SIGTERM:终止某个进程(默认信号)
kill语法:
kill -sig_num PID
kill -sig_name PID
kill %num:杀死后台的某个作业,%不能省略

信号:进程间的通信(IPC)
内存共享等

killall:杀死整个进程树,用法同kill

前台进程:利用终端执行的进程
后台进程:运行执行时不利用终端进行执行的进程

前台-->后台:
ctrl+z:把正在运行的进程调入后台,并停止运行
&:启动时直接调入后台运行

bg:把进程调入后台继续运行
fg:把后台的进程调入前台运行
jobs:查看后台运行的作业
+:下一次将要运行的后台作业

任务计划:调度性的任务

一次性任务计划:
at:设置系统在某一个时间点执行的任务,执行结果以邮件的方式发送给任务发起者
语法:
at data_time
at>at_command1
at>at_command2
at>ctrl+d 保存退出
data_time:时间日期写法
绝对时间:
HH:MM:具体时间点
DD.MM.YY:天.月.年
YY-MM-DD:年-月-日
MM/DD/YY:月/日/年
HH:MM YY-MM-DD:
相对时间:
at now+3minutes:从任务计划创建开始计时,3分钟以后执行

systemctl start atd

```
  # systemctl enable atd
  # systemctl status atd
```
Linux中设置at任务后,会在/var/spool/at/目录生成at任务计划的文件,按照文件名的排序顺序执行

/etc/at.allow和/etc/at.deny:

/etc/at.allow:at任务计划的白名单,仅允许此文件中的用户执行at
/etc/at.deny:at任务计划的黑名单,仅拒绝次文件中的用户执行at

系统先查找/etc/at.allow,再去查找/etc/at.deny,如果两个文件都没有,只有root用户能够执行at

at命令:设置at任务计划
  -l:查看系统中at任务列表,相当于atq
  -r:删除系统中的at任务列表,相当于atrm
  -d:删除系统中的at任务列表,相当于atrm
  -c:列出后续at任务的具体指令内容

batch:用法和at相同,不过batch任务计划是在CPU空闲时执行,当CPU的平均负载小于0.8时执行batch设定任务

```
# batch data_time
  at>at_command1
  at>at_command2
  at>ctrl+d 保存退出
```
    data_time:时间日期写法
    绝对时间:
      HH:MM:具体时间点
      DD.MM.YY:天.月.年
      YY-MM-DD:年-月-日
      MM/DD/YY:月/日/年
      HH:MM YY-MM-DD:
    相对时间:
      at now+3minutes:从任务计划创建开始计时,3分钟以后执行

uptime:查看CPU的平均负载

周期性任务计划:
  系统周期性任务:
    日志轮换:logrotate
    登录文件分析:logwatch任务
    创建locate数据库:updatedb
    创建man帮助信息数据库:CentOS6:makewathis CentOS7:mandb
    RPM软件登录文件的创建
    与网络相关的分析行为:

用户周期性任务:
cron:设置周期任务计划
anacron:当cron周期性任务由于某种原因没有执行,anacron重新执行cron任务

控制crond服务:
systemctl start crond
systemctl enable crond
systemctl status crond

/etc/cron.allow和/etc/cron.deny
/etc/cron.allow:白名单
/etc/cron.deny:黑名单,默认情况下系统只保留/etc/cron.deny

cron相关的配置文件:
/etc/crontab:cron配置文件
/etc/cron.d/:cron辅助配置文件
/var/spool/cron:用户cron任务计划的配置文件,Linux用户设置完cron任务后,会在/var/spool/cron目录下生成以用户名名称的cron文件

crontab:设置/查看/删除等cron任务计划,执行结果以邮件方式发送给任务发起者
-e:编辑cron任务内容
-u:为指定用户设置cron任务计划(只有root用户能够使用)
-l:查看cron任务列表
-r:删除所有的cron列表

设置cron任务语法:
系统设置:
分 时 日 月 周 用户 命令
用户设置:
分 时 日 月 周 命令

时间取值范围:
分钟:0-59
小时:0-23
日期:1-31
月份:1-12或jan,feb,mar,apr ...
周:0-7或sun,mon,tue,wed,thu,fri,sat,其中0和7都代表周日
时间的特殊写法:
通配:*代表任意时间 10 17 * * * cat /etc/fstab
离散时间段:,(逗号) 5,15,30,45,56 17 * * * cat /etc/fstab
连续时间段:-(中横线) 00 23 * * 1-5 cat /etc/fstab
间隔时间段:*/n */5 * * * * cat /etc/fstab 00 10 */5 * * cat /etc/fstab

/etc/crontab文件解析:
SHELL=/bin/bash:cron任务计划执行使用的shell程序
PATH=/sbin:/bin:/usr/sbin:/usr/bin:cron任务计划命令搜索地址
MAILTO=root:cron任务计划执行结果发送给哪个用户
/etc/cron.d:辅助配置文件目录

cron.hourly:cron工具去执行
cron.daily,cron.weekly cron.monthly是有anacron工具去执行

cron设置任务计划时注意事项:
CPU和内存资源集中使用时的问题:
vim /etc/crontab
1,6,11,16,21,26,31,36,41,46,51,56 * * * * root COMMAND1
取消一些不必要的输出:设置命令时可以重定向输出
时间的检验:设置时间时,周与日月不可并存
安全的检验:借有/var/log/cron的内容查看cron任务计划中是否有非法任务
环境变量:系统命令查找和cron任务命令位置由$PATH变量决定,设置cron任务时,使用命令的绝对路径

anacron:当cron周期性任务由于某种原因没有执行,anacron重新执行cron任务

anacron语法:
anacron [options] [job]
-s:依据时间记录文件去判断是否执行
-f:强制执行
-n:立刻运行未执行的任务,不需要延时

anacron执行流程:
1:由/etc/anacrontab分析到cron.daily天数为1天
2:由/var/spool/anacron/cron.daily取出最近一次执行anacron时间戳
3:由2步骤分析出的时间戳和cron任务进行比较,查看哪些cron任务超过1天时间没有执行
4:准备执行指令,根据/etc/anacrontab的设置,将延时执行
5:延时时间过后,执行指令

文件查找:
which:查找命令字所在的位置
locate:模糊匹配(只要包含关键字的文件都查找出来)
不是实时的,基于数据库查找, updatedb升级locate数据库
查找速度特别快

find:查看文件或目录的工具
实时查找
精确查找

遍历整个指定目录中的文件,速度很慢
支持多种查找条件(属主,属组,权限,文件名等)
语法: find [find_path] [条件] [处理动作]
find_path:如果不写,默认在当前工作目录查找
条件:如果不写,会查找出该路径下所有的文件
处理动作:默认打印到屏幕上 print

条件:(匹配标准)
-name file_name:根据文件名进行查找,区分大小写
-iname file_name:根据文件名进行查找,不区分大小写
-regex PATTERN:根据正则表达式的模式进行查找
-user:通过属主进行查找
-group:通过属组进程查找
-uid UID:通过UID号进行查找
-gid GID:通过GID号进行查找
-nouser:查找没有属主的文件
-nogroup:查找没有属组的文件
-type:根据文件类型进行查找
-b:块设备
-c:字符设备
-d:目录
-f:文件
-p:命名管道
-s:套接字文件
-l:链接文件
-size [+ | -] k m g
-size 5k:精确匹配
+:大于
-:小于
根据时间戳进行查找: # stat file_name //查看文件的时间戳
-atime:时间单位为天,access时间
-mtime:时间单位为天,modify时间
-ctime:时间单位为天,change时间
-mmin:时间单位为分钟,modify时间
-amin:时间单位为分钟,access时间
-cmin:时间单位为分钟,change时间
根据权限查找:
-perm MODE:
MODE:644,精确匹配
/MODE:任意一位匹配
-MODE:文件权限能够完全包含此MODE时才能匹配

条件组合:
-a:同时满足多个条件,比如查找属主是tom并且权限为644的文件

-o:满足一个条件即可,比如查找属主是tom或者权限为包含644的文件
-not:不满足条件的被查找出来

出来动作:
-print:默认动作,默认查找结果打印到屏幕
-ls:默认以类似于ls -l的形式显示结果的信息
-ok COMMAND {} \;执行动作时提示用户是否执行,必须以\;结尾,{}代表文件名占位符
-exec COMMAND {} \;执行动作时不提示用户直接执行,必须以\;结尾,{}代表文件名占位符

查找条件通配符:
:通配任意个数的任意字符 # find /etc/ -name "pass" # find /etc/ -name "*pass"
?:通配单个的任意字符 # find /etc/ -name "passw*"

1、查找/var目录下属主为root并且属组为mail的所有文件;
find /var -user root -a -group mail

2、查找/usr目录下不属于root,bin,或student的文件;
find /usr -not -user root -a -not -user bin -a -not -user student
find /usr -not \(-user root -o -user bin -o -user student \)

3、查找/etc目录下最近一周内内容修改过且不属于root及student用户的文件;
find /etc -mtime -7 -not \ (-user root -o -user student \)
find /etc -mtime -7 -not -user root -a -not -user student

4、查找当前系统上没有属主或属组且最近1天内曾被访问过的文件，并将其属主属组均修改为root;
find / \(-nouser -o -nogroup \) -a -atime -1 -exec chown root:root {} \;

5、查找/etc目录下大于1M的文件，并将其文件名写入/tmp/etc.largefiles文件中;
find /etc -size +1M >> /tmp/etc.largefiles

6、查找/etc目录下所有用户都没有写权限的文件，显示出其详细信息;
find /etc -not -perm /222 -ls

进程管理:kernel+process

CPU:ring0:内核空间 ring1-2:库文件等 ring3:用户空间
CPU读取的数据都来自内存
内存:内核空间和用户空间,RAM:易失性存储器

ROM:只读存储器,存储硬件信息等

系统启动流程:CentOS6和CentOS7

PC:Linux

启动过程:
POST(加电自检:PowerOnSelfTest)--BIOS(基本输入输出系统:Basic Input Output System:硬件信息;Boot sequence:启动顺序)--MBR(主引导记录;446字节bootloader)--启动菜单(Linux启动菜单:GRUB)--kernel--initramfs(虚拟根文件系统)--/sbin/init

kernel作用:驱动和测检系统外围硬件或程序
文件系统
安全管理
网络管理
进程管理
驱动程序

内核设计:
单内核:把所有的功能都集成到内核中去
Linux使用单内核,模块化设计 /lib /lib64
/lib/modules/'内核版本号为名称的目录'/kernel
arch:平台架构相关
crypto:安全加密
drivers:驱动程序
fs:文件系统
kernel:内核
lib:内核库 .ko(kernel object)
mm:memory mangement内存管理
net:系统中的TCP/IP协议栈
sound:声卡

modules.dep:解决系统中依赖关系
微内核:把每个功能都做成小模块 Windows solaris

initramfs:CentOS6,虚拟根文件系统,把/proc /sys /dev目录复制到根文件系统
initrd:CentOS5

Linux运行级别:0-6
0：halt关机
1：单用户模式(root, 无须登录), single, 维护模式；
2: 多用户模式，会启动网络功能，但不会启动NFS；维护模式；

3：多用户模式，正常模式；文本界面； CLI:command line interface
4：预留级别；
5：多用户模式，正常模式；图形界面； GUI:graphical user interface
6：重启

chroot:改变根目录
ldd:查看二进制文件所依赖的库文件

1.POST加电自检
POST(PowerOnSelfTest)首先对每一个设备进行检查。完成后会交给BIOS寻找存有引导记录的设备，找到后读入操作系统引导记录，然后将系统控制权交给引导记录，并由引导记录来完成系统的顺利启动。
2.MBR引导
MBR(Master Boot Record) MBR记录一般是在磁盘 0 磁道 1 扇区，共512个字节。前446个字节是BootLoder，后 4*16 的 64 个字节是存放分区信息的，最后 2 个字节是校验信息，一般是 55AA。
3.GRUB（GRand Unified Bootloader）、加载内核
就是MBR中的前 446 个字节，是BooTLoader的一种，它的作用是要选择要启动和加载内核的。
4.kernel
Linux属于单核系统：Kernel+各种外围模块组成。通过grub加载内核后（有可能会借助于ramdisk加载驱动），内核开始自我解压并工作通过虚拟根文件系统initrfs 调用init工作。
ramdisk：内核中的特性之一：使用缓冲和缓存来回事对磁盘上的文件访问；
CentOS 5: initrd，工具程序：mkinitrd
CentOS 6: initramfs， 工具程序：mkinitrd, dracut
5.启动init程序(/sbin/init)
init程序的类型：
SysV: init, CentOS 5
配置文件：/etc/inittab
Upstart: init, CentOS 6
配置文件：/etc/inittab, /etc/init/*.conf
Systemd：systemd, CentOS 7
配置文件：/usr/lib/systemd/system, /etc/systemd/system

启动流程详解:
BootLoader:MBR(位于磁盘中0磁道0柱面1扇区,512字节),后安装的系统的BootLoader覆盖先安装的BootLoader,安装双系统时,先安装Linux,在安装Windows
windows:不允许其他系统引导,也不引导其他系统
linux:
LILO:有众多的限制
GRUB:默认的BootLoader
GRUB引导系统时几个阶段:(配置文件/boot/grub/grub.conf)

default=0 //默认引导系统
timeout=5 //GRUB引导系统超时时间
splashimage=(hd0,0)/grub/splash.xpm.gz //引导系统时图片
hiddenmenu //隐藏菜单
title CentOS 6 (2.6.32-642.el6.x86_64) //系统title及在GRUB中显示的名称
root (hd0,0) //根所在的位置
kernel /vmlinuz-2.6.32-642.el6.x86_64 ro root=UUID=ef361140-6306-423c-90f0-63a93c664f96 rd_NO_LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=auto KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet
initrd /initramfs-2.6.32-642.el6.x86_64.img

Linux系统中内核的名/boot/vmlinuz-2.6.32-642.el6.x86_64

grub配置文件/boot/grub/grub/conf详细解释：
GRUB要求设备名被括在一个括号中。fd表示软盘，hd 表示硬盘（不区分IDE还是SCSI）。其次设备是从0开始编号。分区也是如此，分区和设备之间用一个逗号分开。
default 启动系统时在人为不干预的情况下，默认读取哪一个title，如果安装了多个不同版本内核或者安装了不用的操作系统，会产生多个title，0表示第一个，1第二个，以此类推。
timeout 开机等待用户的超时时间，单位为秒。在超时时间结束时，如果用户没有人为选择，则以default指定的title读取。
hiddenmenu：用于启动时隐藏菜单，除非在timeout之前按下任意键才能看到菜单。
title：定义引导项名称。
root：指定boot分区所在磁盘及分区，如：root (hd0,6)。
kernel：指定kernel文件所在绝对目录地址，如：kernel /vmlinuz-2.6.32-220.el6.x86_64 ，这里的/表示root(hd0,0)分区
initrd：指定ramdisk盘所在绝对目录地址，如：initrd /initramfs-2.6.32-220.el6.x86_64.img
【详解kernel】
kernel /vmlinuz-2.6.32-642.el6.x86_64 ro root=UUID=ef361140-6306-423c-90f0-63a93c664f96 rd_NO_LUKS rd_NO_LVM LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=auto KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM rhgb quiet

kernel /vmlinuz-2.6.32-220.el6.x86_64 ： 制定内核文件的位置
ro：刚开始以只读方式挂载根文件系统
root=UUID=ef361140-6306-423c-90f0-63a93c664f96 ： root根分区设备位置
rd_NO_LUKS LANG=en_US.UTF-8 rd_NO_MD：为了加速引导启动进程，可以指定磁盘是否加密、语言环境、不启用LVM、RAID、键盘等等，节省dracut查找的时间
SYSFONT=latarcyrheb-sun16 KEYTABLE=us：对于有加密磁盘的系统启动，可以指定键盘规格和字体显示等
quiet：启动过程中只有重要信息显示，类似硬件自检的消息不回显示
rhgb：RedHat graphics boot，就是会看到图片来代替启动过程中显示的文本信息，这些信息在启动后用dmesg也可以看到

【注意】
kernel与initrd这两个设置项中，指定的路径都是绝对路径。因为这两个文件都存放在/boot目录。而且/boot所在的分区已经指定，所以就无需再指明kernel与initrd在哪个分区了。如果boot分区为独立分区，那么前面的/boot省略掉。如果boot分区为非独立分区，那么必须加上/boot。
【例如】
独立分区：kernel /vmlinuz-2.6.32-220.el6.x86_64
非独立分区：kernel /boot/vmlinuz-2.6.32-220.el6.x86_64

GRUB分阶段启动:
stage1:第一个阶段,位于BootLoader,为了引导第二阶段
stage1_5:识别文件系统
stage2:加载启动设置等等,读取配置文件/boot/grub/grub.conf

GRUB修复:光盘启动,进入救援模式:
chroot /mnt/sysimage
grub-install --root-directory=/ /dev/sda
grub
grub> root (hd0,0)
grub> setup (hd0)
grub> quit
准备GRUB配置文件
exit
reboot

为GRUB启动菜单添加密码:
grub-crypt --sha-512　//生成加密密码
vim /boot/grub/grub.conf,在hiddenmenu添加一行
password --encryped 生成的密文
为系统添加启动密码:
grub-crypt --sha-512　//生成加密密码
vim /boot/grub/grub.conf,在title中添加一行
password --encryped 生成的密文

Linux内核与内核模块:
内核:/boot/vmlinuz-version
initramfs:/boot/initramfs-kernel_version
内核模块:/lib/modules/kernel_version/kernel
内核源码:/usr/src/kernels/kernel_version
内核版本:/proc/version
系统内核功能:/proc/sys/kernel

Linux关于内核的命令:
lsmod:查看系统中加载的内核模块

结果解释:
module:模块的名称
size:模块的大小
userd by:此模块是否被其他模式使用(依赖关系)

depmod:更新模块的依赖关系并创建依赖关系文件
/lib/modules/kernel_version/modules.dep:解决linux模块依赖关系的文件
-A:查找比/lib/modules/kernel_version/modules.dep新的模块,如果有,才会更新,并写入改文件
-n:不写入依赖关系文件,直接输出到屏幕上
-e:显示当前已经加载的但不可执行的模块名称
modinfo:查看模块的信息
modinfo modules_name

insmod:加载模块
insmod 模块全路径
如:# insmod /lib/modules/2.6.32-642.el6.x86_64/kernel/fs/cifs/cifs.ko
rmmod:删除模块
rmmod 模块名称

modprobe:加载删除模块(不需要模块的完整路径)
-c:列出目前系统中的所有的模块
-l:列出/lib/modules/2.6.32-642.el6.x86_64/kernel当中所有模块的完整文件名
-f:强制加载
-r:删除某个模块,类似于rmmod

Linux内核模块额外的参数设置目录:/etc/modprobe.d/
Linux启动时所用的重要的目录:/etc/sysconfig

init:系统中第一个进程,PID是1,配置文件/etc/inittab
Sys V:init CentOS5:把所有的功能都集成到一个文件中/etc/inittab,启动速度非常慢
Upstart:CentOS6:采用事件驱动(event driven),把功能并行执行

Upstart 概念和术语
Upstart 的基本概念和设计清晰明确。UpStart 主要的概念是 job 和 event。Job 就是一个工作单元，用来完成一件工作，比如启动一个后台服务，或者运行一个配置命令。每个 Job 都等待一个或多个事件，一旦事件发生，upstart 就触发该 job 完成相应的工作。
Job:
Job 就是一个工作的单元，一个任务或者一个服务。可以理解为 sysvinit 中的一个服务脚本。有三种类型的工作：
task job:代表在一定时间内会执行完毕的任务，比如删除一个文件
service job:代表后台服务进程，比如 apache httpd。这里进程一般不会退出，一旦开始运行就成为一个后台精灵进程，由 init 进程管理，如果这类进程退出，由 init 进程重新启动，它们只能由 init 进程发送信号停止。它们的停止一般也是由于所依赖的停止事件而触发的，不过 upstart 也提供命令行工具，让管理人员手动停止某个服务；
Abstract job:仅由 upstart 内部使用，仅对理解 upstart 内部机理有所帮助。我们不用关心它。

除了以上的分类之外，还有另一种工作（Job）分类方法。Upstart 不仅可以用来为整个系统的初始化服务，也可以为每个用户会话（session）的初始化服务。系统的初始化任务就叫做 system job，比如挂载文件系统的任务就是一个 system job；用户会话的初始化服务就叫做 session job。

Job 生命周期:Upstart 为每个工作都维护一个生命周期。一般来说，工作有开始，运行和结束这几种状态。为了更精细地描述工作的变化，Upstart 还引入了一些其它的状态。比如开始就有开始之前(pre-start)，即将开始(starting)和已经开始了(started)种不同的状态，这样可以更加精确地描述工作的当前状态。
工作从某种初始状态开始，逐渐变化，或许要经历其它几种不同的状态，最终进入另外一种状态，形成一个状态机。在这个过程中，当工作的状态即将发生变化的时候，init 进程会发出相应的事件（event）。

Upstart 中 Job 的可能状态
Waiting:初始状态
Starting:Job 即将开始
pre-start:执行 pre-start 段，即任务开始前应该完成的工作
Spawned:准备执行 script 或者 exec 段
post-start:执行 post-start 动作
Running:interim state set after post-start section processed denoting job is running (But it may have no associated PID!)
pre-stop:执行 pre-stop 段
Stopping:interim state set after pre-stop section processed
Killed:任务即将被停止
post-stop:执行 post-stop 段

事件 Event
顾名思义，Event 就是一个事件。事件在 upstart 中以通知消息的形式具体存在。一旦某个事件发生了，Upstart 就向整个系统发送一个消息。没有任何手段阻止事件消息被 upstart 的其它部分知晓，也就是说，事件一旦发生，整个 upstart 系统中所有工作和其它的事件都会得到通知。

Event 可以分为三类: signal，methods 或者 hooks。
Signal 事件是非阻塞的，异步的。发送一个信号之后控制权立即返回。
Methods 事件是阻塞的，同步的。
Hooks 事件是阻塞的，同步的。它介于 Signals 和 Methods 之间，调用发出 Hooks 事件的进程必须等待事件完成才可以得到控制权，但不检查事件是否成功。
事件是个非常抽象的概念，下面我罗列出一些常见的事件，希望可以帮助您进一步了解事件的含义：
系统上电启动，init 进程会发送"start"事件
根文件系统可写时，相应 job 会发送文件系统就绪的事件
一个块设备被发现并初始化完成，发送相应的事件
某个文件系统被挂载，发送相应的事件
类似 atd 和 cron，可以在某个时间点，或者周期的时间点发送事件
另外一个 job 开始或结束时，发送相应的事件
一个磁盘文件被修改时，可以发出相应的事件
一个网络设备被发现时，可以发出相应的事件
缺省路由被添加或删除时，可以发出相应的事件

CentOS6上面Upstart大致的一个启动过程：
1.内核启动init,读取配置文件/etc/inittab,在CentOS6只定义了运行级别
2.系统初始化：（/etc/init/rcS.conf exec /etc/rc.d/rc.sysinit）
3.init找到/etc/inittab文件，确定默认的运行级别(X)（/etc/init/rcS.conf exec telinit $runlevel）
4.触发相应的runlevel事件(/etc/init/rc.conf exec /etc/rc.d/rc $RUNLEVEL)
5.开始运行/etc/rc.d/rc,传入参数X
6./etc/rc.d/rc脚本进行一系列设置，最后运行相应的/etc/rcX.d/中的脚本
7./etc/rcX.d/中的脚本按事先设定的优先级依次启动
8.最后执行/etc/rc.d/rc.local
9.加载终端或X-Window接口

/etc/init:目录定义了系统初始化的配置文件
/proc/cmdline:系统内核设置的参数
/etc/rc.d/rc.sysinit:系统初始化主要配置文件

/etc/rc.d/rc.sysinit 这个文件做了哪些工作：
1、获得网络环境
2、挂载设备
3、开机启动画面Plymouth（取替了过往的 RHGB）
4、判断是否启用SELinux
5、显示于开机过程中的欢迎画面
6、初始化硬件
7、用户自定义模块的加载
8、配置内核的参数
9、设置主机名
10、同步存储器
11、设备映射器及相关的初始化
12、初始化软件磁盘阵列（RAID）
13、初始化 LVM 的文件系统功能
14、检验磁盘文件系统（fsck）
15、磁盘配额(quota)
16、重新以可读写模式挂载系统磁盘
17、更新quota（非必要）
18、启动系统虚拟随机数生成器
19、配置机器（非必要）
20、清除开机过程当中的临时文件
21、创建ICE目录
22、启动交换分区（swap）
23、将开机信息写入/var/log/dmesg文件中

/etc/init/rc.conf:sysv运行级别
/etc/rc.d/存放每个运行级别所需要的服务,比如/etc/rc.d/rc3.d/目录定义的运行级别3所学药的服务
命名: K+数字+服务名
S+数字+服务器
K:kill结束服务
S:start开启服务
数字:代表启动结束的优先级,数字越小优先级越高
/etc/rc.d/rc.local:系统启动加载的最后一个脚本(用户的脚本)

sysv init:chkconfig:定义系统服务在每个级别的运行状态
--list:查看系统中所有服务的状态
chkconfig --level 2345 httpd on
chkconfig --level 2345 httpd off

服务分类:
独立服务:应用程序自己控制
超级服务:依赖于xinetd进程来管理
控制服务:
service service_name start|stop|restart|reload|status
start:启动服务
stop:停止服务
restart:重新启动
reload:重新加载服务
status:查看服务状态

upstart:
initctl start|stop|restart|reload|status service_name

systemd特性
1.平行处理所有服务,加速开机流程:旧的init启动脚本是一项一项任务依序启动的模式,因此不相依的服务也是得要一个一个的等待.systemd可以让所有的服务同时启动,因此系统启动的速度变快了
2.一经要求就回应的on-demand启动方式:systemd全部就是仅有一只systemd服务搭配systemctl指令来处理,无须其他额外的指令来支援.不像systemV还要init,chkconfig,service...等等指令.此外,systemd由于常驻存储器,因此任何要求(on-demand)都可以立即处理后续的daemon启动的任务.
3.服务依赖性的自我检查:由于systemd可以自订服务相依性的检查,因此如果B服务是架构在A服务上面启动的,那当你在没有启动A服务的情况下仅手动启动B服务时,systemd会自动帮你启动A服务!
4.按照daemon功能分类:systemd管理的服务非常多,首先systemd先定义所有的服务为一个服务单位(unit),并将该unit归类到不同的服务类型(type)去.systemd将服务单位(unit)区分为service,socket,target,path,snapshot,timer等多种不同的类型(type),方便管理员的分类与记忆.
5.将多个daemons集合成为一个群组:如同systemV的init里头有个runlevel的特色,systemd亦将许多的功能集合成为一个所谓的target项目,这个项目主要在设计操作环境的建置,所以是集合了许多的daemons,也就是执行某个

target就是执行好多个daemon的意思!
6.向下兼容旧有的init服务脚本:基本上,systemd是可以兼容于init的启动脚本的,因此,旧的init启动脚本也能够通过systemd来管理,只是更进阶的systemd功能就没有办法支援就是了.
7.systemd有些地方无法完全取代init!包括:在runlevel的对应上,大概仅有runlevel1,3,5有对应到systemd的某些target类型而已,没有全部对应;
8.全部的systemd都用systemctl这个管理程序管理,而systemctl支援的语法有限制,不像/etc/init.d/daemon就是纯脚本可以自订参数,systemctl不可自订参数.;
□
Systemctl是一个systemd工具,主要负责控制systemd系统和服务管理器.

Systemd是一个系统管理守护进程、工具和库的集合,用于取代SystemV初始进程.Systemd的功能是用于集中管理和配置类UNIX系统.

在Linux生态系统中,Systemd被部署到了大多数的标准Linux发行版中,只有为数不多的几个发行版尚未部署.Systemd通常是所有其它守护进程的父进程,但并非总是如此.

CentOS7启动流程:

lsinitrd /boot/initramfs-3.10.0-514.el7.x86_64.img //查看虚根文件系统

在内核载入完毕、进行完硬件侦测与驱动程序载入后，内核会启动第一个进程systemd, systemd 最主要的功能就是准备软件执行的环境，包括系统的主机名称、网络设定、语言设置、文件系统及其他服务的启动。 而所有的动作都会通过 systemd 的预设启动服务集合/etc/systemd/system/default.target设定。 另外， systemd 已经舍弃沿用多年的 system V 的 runlevel!
□常见的操作环境 target 与兼容于 runlevel 的等级可以作为预设的操作环境 (default.target) 的主要项目有：multi-user.target 以及 graphical.target 这两个。当然还有rescue.target, emergency.target, shutdown.target 等等，以及 initrd.target！但是过去的 systemV 使用的是一个称为 runlevel (执行等级) 的概念来启动系统的，systemd 为了兼容于旧式的 systemV 操作行为， 所以也将 runlevel 与操作环境做个结合！你可以使用底下的方式来查询两者间的对应关系

ll -d /usr/lib/systemd/system/runlevel*.target
yum -y install bash-completion //补齐参数或选项

systemctl list-dependencies graphical.target //查看启动依赖关系
systemctl list-dependencies mutli-user.target //查看字符界面启动依赖关系

sysinit.target完成的功能：
1.特殊文件系统装置的挂载：包括 dev-hugepages.mount dev-mqueue.mount 等挂载服务，主要在挂载跟巨量存储器分页使用与讯息队列的功能。 挂载成功后，会在 /dev 底下建立 /dev/hugepages/, /dev/mqueue/ 等目录;
2.特殊文件系统的启用：包括磁碟阵列、网络磁碟 (iscsi)、LVM 档案系统、档案系统对照服务 (multipath) 等等，也会在这里被侦测与使用到!
3.开机过程的讯息传递与动画执行：使用 plymouthd 服务搭配 plymouth 指令来传递动画与讯息
4.日志式登录档的使用：就是 systemd-journald 这个服务的启用啊!

5.载入额外的核心模块：透过 /etc/modules-load.d/*.conf 档案的设定，让核心额外载入管理员所需要的核心模块！
6.载入额外的核心参数设定：包括 /etc/sysctl.conf 以及 /etc/sysctl.d/*.conf 内部设定！
7.启动系统的随机数产生器：随机数产生器可以帮助系统进行一些密码加密演算的功能
8.设定终端机 (console) 字形
9.启动动态装置管理员：就是udev

basic.target完整的功能：
1.载入 alsa 音效驱动程序；
2.载入 firewalld 防火墙：CentOS 7.x 以后使用 firewalld 取代 iptables 的防火墙设定，虽然最终都是使用 iptables 的架构， 不过在设定上面差很多。
3.载入 CPU 的微指令功能；
4.启动与设定 SELinux 的安全上下文
5.将目前的开机过程所产生的开机信息写入到 /var/log/dmesg 当中
6.由 /etc/sysconfig/modules/*.modules 及 /etc/rc.modules 载入管理员指定的模块！
7.载入 systemd 的 timer 功能；

systemd 启动 multi-user.target 下的服务:
启动服务设定的脚本目录:
□/usr/lib/systemd/system (系统预设的服务启动脚本设定)
□/etc/systemd/system (管理员自己开发与设定的脚本设定)
而使用者针对主机的本机服务与服务器网络服务的各项 unit 若要 enable 的话，就是将它放到 /etc/systemd/system/multi-user.target.wants/ 这个目录底下做个链接，这样就可以在开机的时候去启动他。

与sys V兼容的rc.local操作:

```
# chmod +x /etc/rc.d/rc.local
# systemctl start rc-local
# systemctl enable rc-local
```

unit分类:
.service:一般服务类型(serviceunit):主要是系统服务,包括服务器本身所需要的本机服务以及网络服务
.socket:套接字(socketunit):主要是IPC(Inter-processcommunication)的传输讯息(socketfile)功能.这种类型的服务通常在监控讯息传递的套接字,当有透过此套接字传递讯息来说要连结服务时,就依据当时的状态将该用户的要求传送到对应的daemon,若daemon尚未启动,则启动该daemon后再传送用户的要求.
.target:执行环境类型(targetunit)
.mount:文件系统挂载相关的服务(automountunit/mountunit):例如来自网络的自动挂载、NFS档案系统挂载等与档案系统相关性较高的程序管理.
.path:侦测特定档案或目录类型(pathunit):某些服务需要侦测某些特定的目录来提供队列服务,例如最常见的打印服务,就是透过侦测打印队列目录来启动打印功能!
.timer:循环执行的服务(timerunit):这个东西有点类似anacrontab!不过是由systemd主动提供的,比anacrontab更加有弹性!

与systemd的daemon运作过程相关的目录简介:
1;/usr/lib/systemd/system/:使用CentOS官方提供的软件安装后,预设的启动脚本设定档都放在这里
2./run/systemd/system/:系统执行过程中所产生的服务脚本,这些脚本的优先序要比/usr/lib/systemd/system/高!
3./etc/systemd/system/:管理员依据主机系统的需求所建立的执行脚本,其实这个目录有点像以前/etc/rc.d/rc5.d/Sxx之类的功能!执行优先序又比/run/systemd/system/高!
4./etc/sysconfig/*:几乎所有的服务都会将初始化的一些选项设定写入到这个目录下,举例来说,mandb所要更新的manpage索引中,需要加入的参数就写入到此目录下的man-db当中!而网络的设定则写在/etc/sysconfig/network-scripts/这个目录内.
5./var/lib/:一些会产生资料的服务都会将他的资料写入到/var/lib/目录中.举例来说,数据库管理系统Mariadb的数据库预设就是写入/var/lib/mysql/这个目录下!
6./run/:放置了好多daemon的暂存档,包括lockfile以及PIDfile等等.

systemctl项目简介:
#cat /usr/lib/systemd/system/sshd.service
.service大概能够将整个设定分为三个部份,就是:
□[Unit]:unit本身的说明,以及与其他相依赖daemon的设定,包括在什么服务之后才启动此unit之类的设定值;
□[Service],[Socket],[Timer],[Mount],[Path]..:不同的unittype就得要使用相对应的设定项目.我们拿的是sshd.service来当模板,所以这边就使用Service来设定.这个项目内主要在规范服务启动的脚本、环境设定档档名、重新启动的方式等等.
□[Install]:这个项目就是将此unit安装到哪个target里面去的意思!
至于设定档内有些设定规则还是得要说明一下:
□设定项目通常是可以重复的,例如我可以重复设定两个After在设定档中,不过,后面的设定会取代前面的!因此,如果你想要将设定值归零,可以使用类似『After=』的设定,亦即该项目的等号后面什么都没有,就将该设定归零了(reset).
□如果设定参数需要有『是/否』的项目(布尔值,boolean),你可以使用1,yes,true,on代表启动,用0,no,false,off代表关闭!
空白行、开头为#或;的那一行,都代表注解!

Unit部分说明:
Description:服务的描述信息
Documentation:提供给管理员的一些帮助文档!提供的文件可以是如下的格式:□Documentation=http://www....□Documentation=man:sshd(8)Documentation=file:/etc/ssh/sshd_config
After:说明此unit是在哪个daemon启动之后才启动的意思!基本上仅是说明服务启动的顺序而已,并没有强制要求里头的服务一定要启动后此unit才能启动.以sshd.service的内容为例,该服务提到After后面有network.target以及sshd-keygen.service,但是若这两个unit没有启动而强制启动sshd.service的话,那么sshd.service应该还是能够启动的!这与Requires的设定是有差异的!
Before:与After的意义相反,是在什么服务启动前最好启动这个服务的意思.不过这仅是规范服务启动的顺序,并非强制要求的意思.
Requires:明确的定义此unit需要在哪个daemon启动后才能够启动!就是设定依赖服务啦!如果在此项设定的前面服务没有启动,那么此unit就不会被启动!
Wants:与Requires刚好相反,规定的是这个unit之后最好还要启动什么服务比较好的意思!不过,并没有明确的规定就是了!主要的目的是希望建立让使用者比较好操作的环境.因此,这个Wants后面接的服务如果没有启动,其实不会影响到这个unit本身!
Conflicts:代表冲突的服务!也就是这个项目后面接的服务如果有启动,那么我们这个unit本身就不能启动!我们unit有启动,则此项目后的服务就不能启动!

[Service]部分说明:
Type:说明这个daemon启动的方式,会影响到ExecStart!一般来说,有底下几种类型
□simple:预设值,这个daemon主要由ExecStart接的指令串来启动,启动后常驻于存储器中.
□forking:由ExecStart启动的程序透过spawns延伸出其他子程序来作为此daemon的主要服务.原生的父程序在启动结束后就会终止运作.传统的unit服务大多属于这种项目,例如httpd这个WWW服务,当httpd的程序因为运作过久因此即将终结了,则systemd会再重新生出另一个子程序持续运作后,再将父程序删除.
□oneshot:与simple类似,不过这个程序在工作完毕后就结束了,不会常驻在存储器中.
□dbus:与simple类似,但这个daemon必须要在取得一个D-Bus的名称后,才会继续运作!　　因此设定这个项目时,通常也要设定BusName=才行!
□idle:与simple类似,意思是,要执行这个daemon必须要所有的工作都顺利执行完毕后才会执　行.这类的daemon通常是开机到最后才执行即可的服务!
比较重要的项目大概是 simple, forking 与 oneshot 了！毕竟很多服务需要子程序 (forking)，而有更多的动作只需要在开机的时候执行一次(oneshot)，例如文件系统的检查与挂载啊等等的。

EnvironmentFile:可以指定启动脚本的shell环境！例如 sshd.service 的设定档写入到 /etc/sysconfig/sshd 当中！你也可以使用 Environment= 后面接多个不同的 Shell 变量来给予设定！

ExecStart:就是实际执行此 daemon 的指令或脚本程序。你也可以使用 ExecStartPre (之前) 以及 ExecStartPost (之后) 两个设定项目来在实际启动服务前，进行额外的指令行为。 但需要特别注意的是，指令串仅接受『指令 参数 参数...』的格式，不能接受 <, >, >>, |, & 等特殊字符，很多的 bash 语法也不支援喔！ 所以，要使用这些特殊的字符时，最好直接写入到指令脚本里面去！不过，上述的语法也不是完全不能用，亦即，若要支援比较完整的 bash 语法，那你得要使用 Type=oneshot 才行喔！ 其他的 Type 才不支持这些字符。

ExecStop:与 systemctl stop 的执行有关，关闭此服务时所进行的指令。

Restart:当设定 Restart=1 时，则当此 daemon 服务终止后，会再次的启动此服务。举例来说，如果你在 tty2 使用文字界面登入，操作完毕后登出，基本上，这个时候 tty2 就已经结束服务了。 但是你会看到荧幕又立刻产生一个新的 tty2 的登入画面等待你的登入！那就是 Restart 的功能！除非使用 systemctl 强制将此服务关闭，否则这个服务会源源不绝的一直重复产生！

RemainAfterExit:当设定为 RemainAfterExit=1 时，则当这个 daemon 所属的所有程序都终止之后，此服务会再尝试启动。这对于 Type=oneshot 的服务很有帮助！

TimeoutSec:若这个服务在启动或者是关闭时，因为某些缘故导致无法顺利『正常启动或正常结束』的情况下，则我们要等多久才进入『强制结束』的状态！

KillMode:可以是 process, control-group, none 的其中一种，如果是 process 则 daemon 终止时，只会终止主要的程序 (ExecStart 接的后面那串指令)，如果是 control-group 时， 则由此 daemon 所产生的其他 control-group 的程序，也都会被关闭。如果是 none 的话，则没有程序会被关闭喔！

RestartSec:与 Restart 有点相关性，如果这个服务被关闭，然后需要重新启动时，大概要 sleep 多少时间再重新启动的意思。预设是 100ms (毫秒)。

Install部分说明:
WantedBy:这个设定后面接的大部分是 *.target unit ！意思是，这个 unit 本身是附挂在哪一个 target unit 底下的！一般来说，大多的服务性质的 unit 都是附挂在 multi-user.target 底下！

Also:当目前这个 unit 本身被 enable 时，Also 后面接的 unit 也请 enable 的意思！也就是具有相依性的服务可以写在这里呢！

Alias:进行一个连结的别名的意思！当 systemctl enable 相关的服务时，则此服务会进行连结档的建立！以 multi-user.target 为例，这个家伙是用来作为预设操作环境 default.target 的规划， 因此当你设定用成 default.target 时，这个 /etc/systemd/system/default.target 就会链接到 /usr/lib/systemd/system/multi-user.target ！

systemctl命令详细使用说明:

1.首先检查你的系统中是否安装有systemd并确定当前安装的版本

#systemd--version

2.检查systemd和systemctl的二进制文件和库文件的安装位置

whereis systemd
whereis systemctl

3.检查systemd是否运行

ps-eaf | grepsystemd
注意:systemd是作为父进程（PID=1）运行的.在上面带（-e）参数的ps命令输出中,选择所有进程,（-a）选择除会话前导外的所有进程,并使用（-f）参数输出完整格式列表（即-eaf）.

4.分析systemd启动进程

systemd-analyze

5.分析启动时各个进程花费的时间

systemd-analyze blame

6.分析启动时的关键链

systemd-analyze critical-chain

7.列出所有可用单元

systemctl list-unit-files

8.列出所有运行中单元

systemctl list-units

9.列出所有失败单元

systemctl --failed

10.检查某个单元（如cron.service）是否开机自启

#systemctl is-enabled crond.service

11.检查某个单元或服务是否运行

systemctl status firewalld.service

使用Systemctl控制并管理服务

12.列出所有服务（包括启用的和禁用的）

#systemctl list-unit-files --type=service

13.Linux中如何启动、重启、停止、重载服务以及检查服务（如httpd.service）状态

#systemctl start httpd.service
#systemctl restart httpd.service
#systemctl stop httpd.service
#systemctl reload httpd.service
#systemctl status httpd.service

注意:当我们使用systemctl的start,restart,stop和reload命令时,我们不会从终端获取到任何输出内容,只有status命令可以打印输出.

14.如何激活服务并在启动时启用或禁用服务（即系统启动时自动启动服务）

systemctl is-active httpd.service
systemctl enable httpd.service
systemctl disable httpd.service

15.如何屏蔽（让它不能启动）或显示服务（如httpd.service）

#systemctl mask httpd.service

#systemctl unmask httpd.service

16.使用systemctl命令杀死服务

systemctl kill httpd
systemctl statu shttpd

使用Systemctl控制并管理挂载点

17.列出所有系统挂载点

#systemctl list-unit-files --type=mount

18.挂载、卸载、重新挂载、重载系统挂载点并检查系统中挂载点状态

#systemctl start tmp.mount
#systemctl stop tmp.mount
#systemctl restart tmp.mount
#systemctl reload tmp.mount
#systemctl status tmp.mount

19.在启动时激活、启用或禁用挂载点（系统启动时自动挂载）

#systemctl is-active tmp.mount
#systemctl enable tmp.mount
#systemctl disable tmp.mount

20.在Linux中屏蔽（让它不能启用）或可见挂载点

systemctl mask tmp.mount
#s ystemctl unmask tmp.mount

使用Systemctl控制并管理套接口

21.列出所有可用系统套接口

#systemctl list-unit-files --type=socket

22.在Linux中启动、重启、停止、重载套接口并检查其状态

#systemctl start cups.socket
#systemctl restart cups.socket
#systemctl stop cups.socket
#systemctl reload cups.socket
#systemctl status cups.socket

23.在启动时激活套接口,并启用或禁用它（系统启动时自启动）

#systemctl is-active cups.socket
#systemctl enable cups.socket
#systemctl disable cups.socket

24.屏蔽（使它不能启动）或显示套接口

#systemctl mask cups.socket
#systemctl unmask cups.socket

服务的CPU利用率（分配额）

25.获取当前某个服务的CPU分配额（如httpd）

systemctl show -p CPUShares httpd.service

注意:各个服务的默认CPU分配份额=1024,你可以增加/减少某个进程的CPU分配份额.

26.将某个服务（httpd.service）的CPU分配份额限制为2000CPUShares/

#systemctl set-property httpd.service CPUShares=2000
#systemctl show -p CPUShares httpd.service
CPUShares=2000
注意:当你为某个服务设置CPUShares,会自动创建一个以服务名命名的目录（如httpd.service）,里面包含了一个名为90-CPUShares.conf的文件,该文件含有CPUShare限制信息,你可以通过以下方式查看该文件:

vim /etc/systemd/system/httpd.service.d/50-CPUShares.conf
[Service]
CPUShares=2000

27.检查某个服务的所有配置细节

#systemctl show httpd

28.分析某个服务（httpd）的关键链

#systemd-analyze critical-chain httpd.service

29.获取某个服务（httpd）的依赖性列表

#systemctl list-dependencies httpd.service

30.按等级列出控制组

systemd-cgls

31.按CPU、内存、输入和输出列出控制组

systemd-cgtop

控制系统运行等级

32.启动系统救援模式

#systemctl rescue

33.进入紧急模式

systemctl emergency

34.列出当前使用的运行等级

systemctl get-default

35.启动运行等级5,即图形模式

#systemctl isolate runlevel5.target
或
#systemctl isolate graphical.target

36.启动运行等级3,即多用户模式（命令行）

#systemctl isolate runlevel3.target
或
#systemctl isolate multiuser.target

36.设置多用户模式或图形模式为默认运行等级

#systemctl set-default runlevel3.target
#systemctl set-default runlevel5.target

37.重启、停止、挂起、休眠系统或使系统进入混合睡眠

#systemctl reboot
#systemctl halt
#systemctl suspend
#systemctl hibernate
#systemctl hybrid-sleep