Linux进程管理和任务计划

一.进程相关概念介绍

1.操作系统内核的作用

1.1操作系统

- 操作系统一般具有下面两种意思：
  - 表示包括管理计算机资源的核心软件和附带的标准软件工具，附带的标准软件如：命令行解释器、 GUI用户图形界面、文件工具及文编辑器等等。
  - 更加狭义的维度上，只表示管理和分配计算机资源的核心软件。(管理计算机资源如：CPU管理、RAM内存管理和设备管理等)
- 大多数人所说的操作系统一词一般指第一种意思，包括了附带的标准软件。

1.2内核

- 内核一般被认为是只表示管理和分配计算机资源的核心软件。是操作系统的核心部分。
- 内核主要负责以下工作

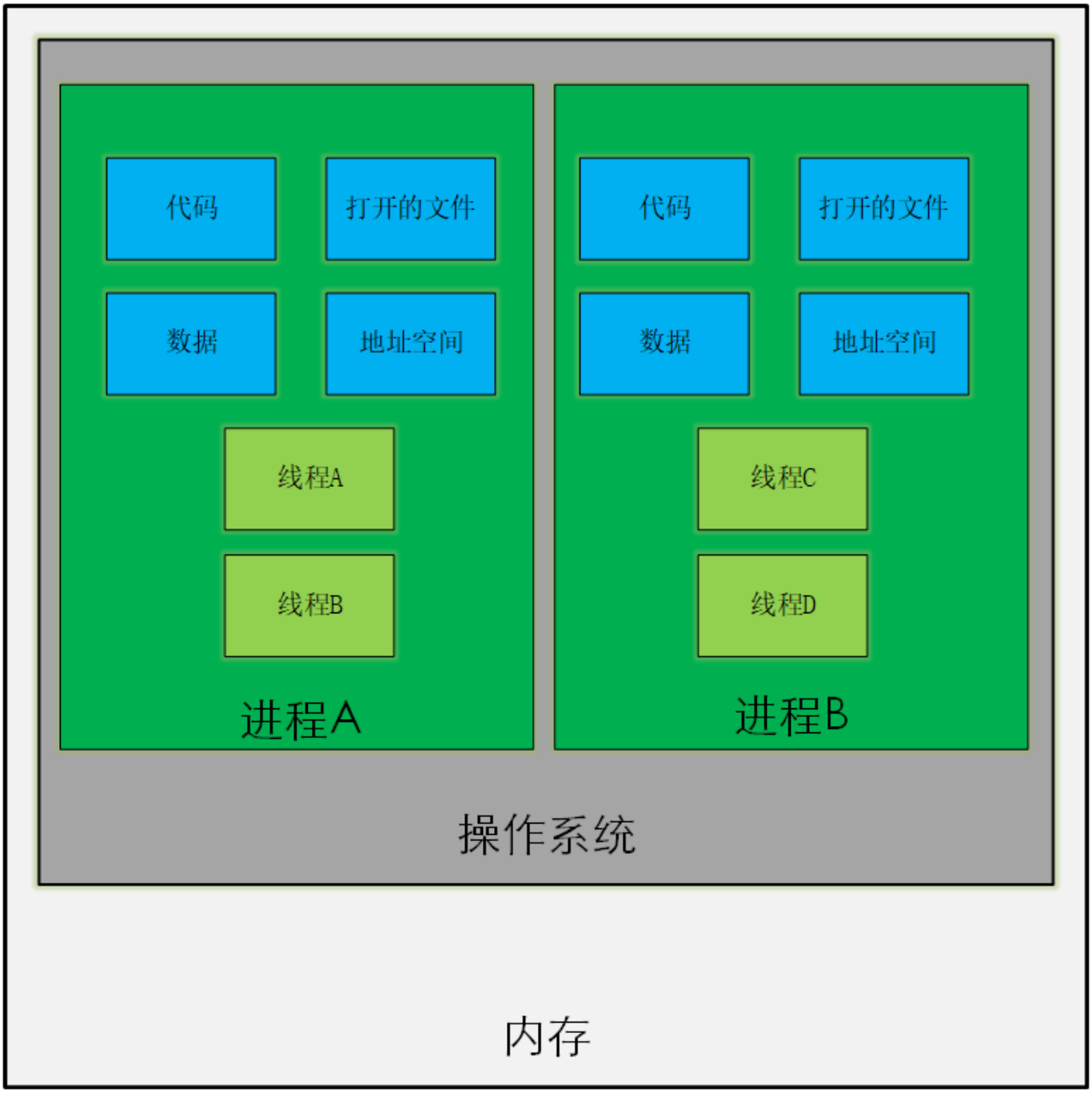
内核功能	解释
进程调度	现在的计算机一般会有多个物理核心用来执行程序的指令。Linux系统是一个抢占式多任务(preemptive multitasking)的操作系统,多任务就意味着多个进程可以同时驻留在内存中得到CPU的处理，抢占式就意味着是哪个进程使用CPU或者使用多久的规则由内核进程调度器决定而不是CPU自己决定处理哪个进程
内存管理	目前内存已近越来越大，但是软件体积也同样在增长；计算机的物理内存仍然是比较稀缺的资源，这就需要内核合理的管理分配内存给系统的各个进程
提供文件系统功能	内核能够在磁盘上创建特定文件系统，以允许文件被创建，复制更新和删除等
创建和销毁进程	内核可以加载某个程序到内存，并为其分配其运行所需要的资源(CPU、内存、对文件的访问等)，在内存中运行的某个程序的实例就叫做进程。当程序被内核销毁或者自己运行结束后，内核还需要保证其使用的资源被释放以便于后续进程重用
访问设备	连接到计算机上的设备(如：麦克风、显示器、键盘鼠标、磁盘和磁带、软盘等)是的计算机内部和计算机之间及计算机和外部世界可以进行信息交互，允许计算机进行输入、输出等操作。内核为每个上层应用程序提供了一个标准的接口来访问设备，同时仲裁多个进程对设备的访问
提供网络功能	内核代替用户进程收发网络数据。

内核功能	解释
提供系统调用应用程序编程接口	进程可以通过一个入口请求内核完成多种操作，该入口就是系统调用
提供虚拟个人电脑抽象功能	在linux系统，多个用户可以同时登陆系统并进行不同的操作，内核提供了一个抽象层来保证各个用户的操作和对设备的使用的隔离性

2.程序？进程？线程？

程序、进程、线程		
程序		一般程序以两种形式存在：源代码的形式和二进制可执行程序的形式。源代码是人类可以读的文本(包括一系列的使用列如C语言编写的语句)。程序要被CPU执行，必须编译链接为计算机可以识别的二进制机器码指令(windows内的.exe文件；linux下的.o文件等)；二者被认为是同义词，都代表程序。
进程	简单的说，一个进程就是一个正在执行的程序的实例。当一个程序被执行时，内核会将该程序的指令加载进内存，分配内存空间给程序变量并设置相应的数据结构来记录该进程的信息(Linux内核对该种数据结构的实现叫task list，保存有进程ID、退出状态、用户ID和组ID等)。因此，进程也可以描述为：运行的程序及其包括的数据和操作。	
线程	在现代的UNIX实现版中，每个进程可以有多个线程运行。一个理解线程比较好的表述是：共享同一块内存空间和其他属性的轻量级进程的集合。每个线程都执行同一个代码块并共享相同的数据区域和堆。然而，每个线程都有自己的栈空间(包含本地变量和函数调用连接信息)。线程之间可以通过其共享的全局变量进行通讯，进程亦可以通过IPC和同步机制进行通讯。	

- 进程图示



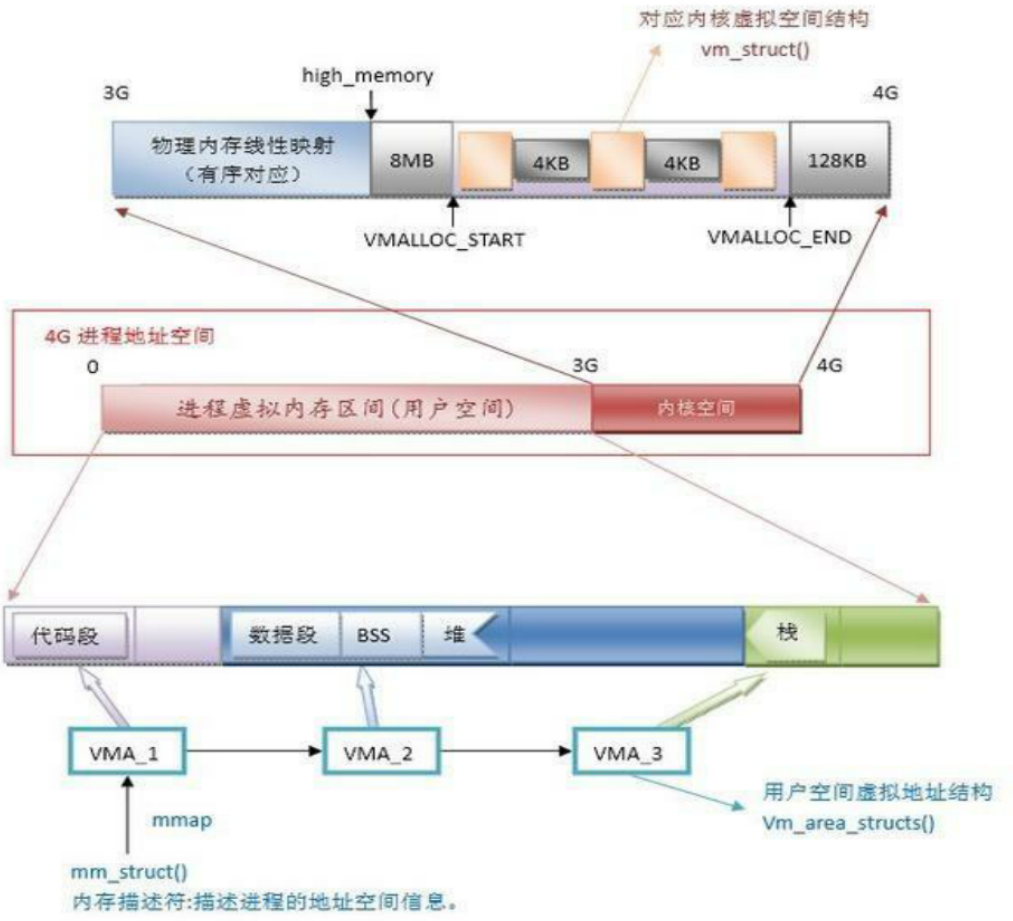
3.Linux中的进程

- task struct：Linux内核存储进程信息的数据结构格式
- task list：多个任务的的task struct组成的链表
- Linux进程的创建
    - init：第一个进程为系统启动时创建的最重要的进程，其进程ID为1
    - 除了init的其他进程：都由其父进程创建，使用fork()系统调用创建。
- 查看线程：cat /proc/PID/status |grep -i threads
- Page Frame: 页框，用存储页面数据，存储Page 4k
- 物理地址空间和线性地址空间

- MMU：Memory Management Unit 负责转换线性和物理地址的硬件
- TLB：Translation Lookaside Buffer 翻译后备缓冲器 用于保存虚拟地址和物理地址映射关系的缓存,便于快速访问物理页

3.用户模式和内核模式概念

- 现代处理器架构一般都允许CPU在至少两个不同模式运行：用户模式(user mode)和内核模式 (kernel mode：内核模式有时候也被称为监管模式)，CPU自带的硬件指令允许在两个模式之间切换。 相应地，虚拟内存的某部分可被标记为用户空间(user space)或内核空间(kernel space)。当在 用户模式运行某进程时CPU只可以访问标记为用户控件的内存区域，尝试访问被标记为内核空间的内存 将会导致一个硬件异常(hardware exception)错误。
- 用户空间和内核空间

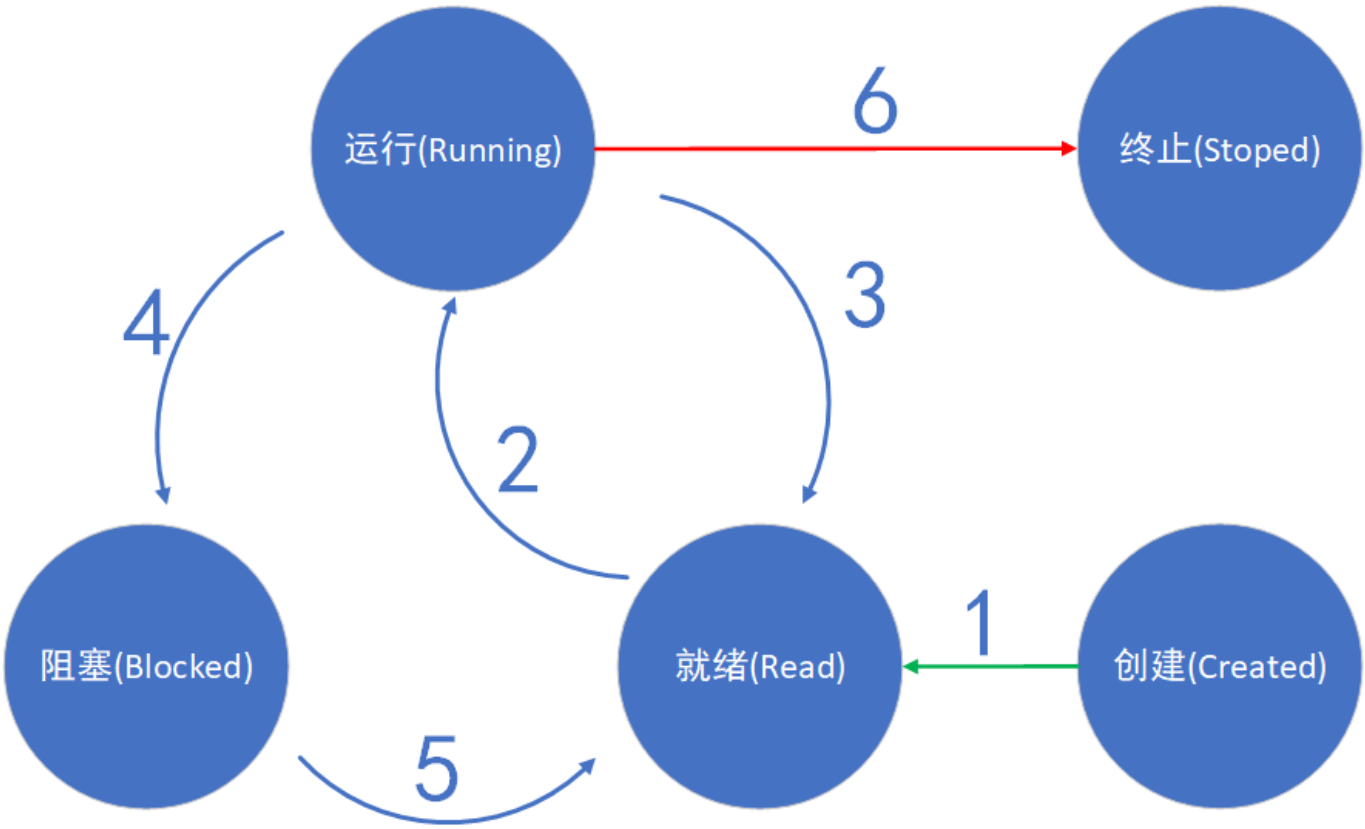


- 用户模式和内核模式简单对比

用户模式	内核模式
不能执行关机指令	执行关机指令来关闭当前系统
不可以访问内存管理硬件	可以访问用于内存管理的硬件
不能	初始化设备I/O操作
不能	执行大多数特权指令

4.进程的状态及状态之间的转换

- 在进程的整个生命周期中可能会经过一下状态：创建、运行、阻塞、就绪、终止。
- 图示如下：



1：进程创建获取相关资源转换为就绪状态
2：调度器调度该进程
3：时间片用完，调度器调度另一个进程
4：该进程发出I/O请求，被阻塞等待I/O完成
5：I/O操作完成，转换为就绪状态
6：进程作业完成释放资源，终止

- 各状态说明

创建状态	进程在创建时需要申请一个空白PCB(process control block进程控 制块)，向其中填写控制和管理进程的信息，完成资源分配。如果创建工作无法完 成，比如资源无法满足，就无法被调度运行，把此时进程所处状态称为创建状态
就绪状态	进程已准备好，已分配到所需资源，只要分配到CPU就能够立即运行

执行状态	进程处于就绪状态被调度后，进程进入执行状态
阻塞状态	正在执行的进程由于某些事件（I/O请求，申请缓存区失败）而暂时无法运行进程受到阻塞。在满足请求时进入就绪状态等待系统调用
终止状态	进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行

- 状态之间转换的四种情况

运行to就绪	1，主要是进程占用CPU的时间过长，而系统分配给该进程占用CPU的时间是有限的；2，在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行时，该进程就被迫让出CPU，该进程便由执行状态转变为就绪状态
就绪to运行	运行的进程的时间片用完，调度就转到就绪队列中选择合适的进程分配CPU
运行to阻塞	正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如发生了I/O请求
阻塞to就绪	进程所等待的事件已经发生，就进入就绪队列

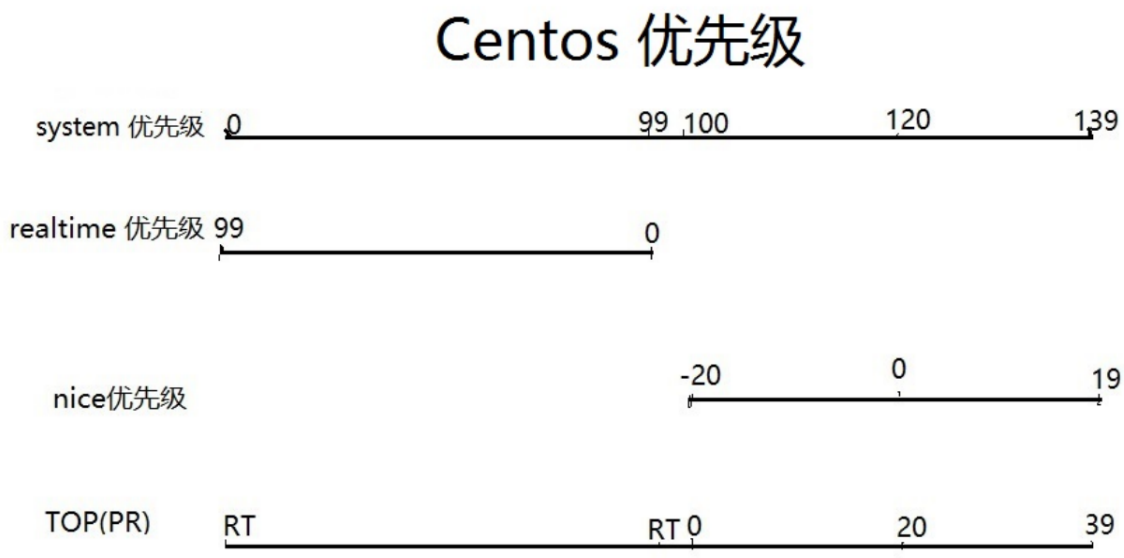
5.Linux的进程状态

- linux进程类型： 守护进程: daemon,在系统引导过程中启动的进程，和终端无关进程 前台进程：跟终端相关，通过终端启动的进程 注意：两者可相互转化
- linux进程状态： 运行态：running 就绪态：ready 睡眠态： 可中断睡眠态：interruptable 不可中断睡眠态：uninterruptable 停止态：stopped,暂停于内存，但不会被调度，除非手动启动 僵死态：zombie，结

束进程，父进程结束前，子进程不关闭

6.Linux的进程优先级

- centos进程优先级



- 系统优先级：数字越小，优先级越高
- 0-139：每个优先级有140个运行队列和过期队列
- 实时优先级: 99-0 值最大优先级最高
- nice值：-20到19，对应系统优先级100-139

6.进程间通信IPC

- IPC进程间通信(Inter Process Communication)
- 同一主机：pipe

```
管道
    socket   套接字文件
    signal   信号
    shm
shared memory
    semaphore
信号量，一种计数器
```

- 不同主机：socket IP和端口号

```
    RPC
    remote procedure call
    MQ
消息队列，如：Kafka，RabbitMQ，ActiveMQ
```

二.Linux下进程相关的工具介绍

1.pstree命令

- patree命令以树状结构显示当前系统进程
- 用法

```
SYNOPSIS
       pstree [-a, --arguments] [-c, --compact] [-h, --highlight-all, -Hpid, --
high-
       light-pid pid] [-g] --show-pgids] [-l, --long] [-n, --numeric-sort]
       [-N, --ns-sortns [-p, --show-pids] [-s, --show-parents] [-S, --ns-changes]
       [-t, --thread-names] [-T, --hide-threads] [-u, --uid-changes] [-Z, --
security-con-
       text] [-A, --ascii, -G, --vt100, -U, --unicode] [pid, user]
       pstree -V, --version
    -p 显示每个进程的进程ID
    -a 显示进程关联的命令及其所附带的运行参数，如例1
    -A 使用ASCII码值画出树干,显示内容不变
    -c 禁用相同的子进程折叠，pstree显示时默认折叠子进程
    -h 高亮显示当前进程和其父进程，如果终端不支持高亮则不作任何操作，如例2
    -H 高亮所指定的进程号所代表的进程，如果使用该选项而未指定pid，则pstree执行失败，如例
3
    -g 显示进程组ID号(PGID),如果PID也指定，那就先显示进程PID，格式(PID,PGID)
    -l 使用长格式
    -n 将同一个父进程的子进程按照进程ID大小从小到大排序显示
    -s 显示所指定的进程的父进程，如例4
    -t 如果可行就显示每个进程的线程全名,pstree默认将线程放在[{...}]内。
    -T 只显示进程，忽略线程
```

- 例1

```
[root@centos8 ~]#pstree -a
systemd --switched-root --system --deserialize 17
  ├─ModemManager
  │    └─2*[{ModemManager}]
  ├─NetworkManager --no-daemon
  │    └─2*[{NetworkManager}]
  ├─VGAuthService -s
  ├─abrt-dbus -t133
  │    └─2*[{abrt-dbus}]
  ├─abrt-dump-journ -fxtD
  ├─abrt-dump-journ -fxtD
  ├─abrtd -d -s
  │    └─2*[{abrtd}]
  ├─accounts-daemon
```

```
  |   └─2*[{accounts-daemon}]
  ├─alsactl -s -n 19 -c -E ALSA_CONFIG_PATH=/etc/alsa/alsactl.conf--
initfile=/lib/als
  ├─atd -f
  ...
[root@centos8 ~]#pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─VGAuthService
        ├─abrt-dbus───2*[{abrt-dbus}]
        ├─2*[abrt-dump-journ]
        ├─abrtd───2*[{abrtd}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─alsactl
        ├─atd
        ├─auditd─┬─sedispatch
        |        └─2*[{auditd}]
```

- 例2

```
[root@centos8 ~]#pstree -ha
...
├─systemd-udevd
  ├─tmux: server new -s class    # 高亮
  |   ├─bash
  |   |   └─man pstree
  |   |       └─less
  |   ├─bash
  |   |   └─ssh 172.20.3.82
  |   └─bash                     # 高亮
  |       └─pstree -ha           # 高亮
  ...
```

- 例3

```
[root@centos8 ~]#pstree -H48708 -p
      ...
      ├─systemd-udevd(668)
           ├─`tmux: server(48708)`─┬─bash(48709)───man(49846)───less(49858)      #
反引号内的进程高亮
           |                       ├─bash(48744)───ssh(48782)
           |                       └─bash(49899)───pstree(49980)
           ...
```

- 例4

```
[root@centos8 ~]#pstree -s 1316 -p
systemd(1)——tuned(1022)——{tuned}(1316)
```

2.ps命令

- ps是linux系统中查看进程的有力工具之一。man帮助指明其用于报告当前系统所有进程的一个快照
- ps支持多种风格的命令选项
  - UNIX选项 如-A -e
  - BSD选项 如a
  - GNU选项 如--help
- ps用法:不带选项默认显示当前终端中的进程

```
BSD风格：
    a 选项包括所有终端中的进程
    x 选项包括不链接终端的进程
    u 选项显示进程所有者的信息
    f 选项显示进程树,相当于 --forest
    k|--sort  属性 对属性排序,属性前加- 表示倒序
    o 属性… 选项显示定制的信息 pid、cmd、%cpu、%mem、ni、pri、rtpri、psr(CPU编号)
    L 显示支持的属性列表
```

```
UNIX风格选项：
    -C cmdlist 指定命令，多个命令用，分隔
    -L 显示线程
    -e 显示所有进程，相当于-A
    -f 显示完整格式程序信息
    -F 显示更完整格式的进程信息
    -H 以进程层级格式显示进程相关信息
    -u userlist  指定有效的用户ID或名称
    -U userlist 指定真正的用户ID或名称
    -g gid或groupname  指定有效的gid或组名称
    -G gid或groupname  指定真正的gid或组名称
    -p pid 显示指pid的进程
    --ppid pid  显示属于pid的子进程
    -t  ttylist  指定tty,相当于 t
    -M  显示SELinux信息，相当于Z
```

- 值得注意的是，下面的两个命令是不一样的，虽然其结果表现的几乎一样：

> ps -aux ps aux

- POSIX和unix标准需要使用-aux选项来指明所有属于用户x的进程。而BSD风格的x选项指明包括不链接终端的进程,如下面的例子：

```
ps -auroot              # 该命令将会列出所有属于root用户的进程

ps auroot               # 该命令出错
```

- ps输出选项参数介绍

```
[root@centos8 ~]#ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.5 242356  8332 ?        Ss   00:58   0:07
/usr/lib/systemd/systemd --switched-root --system --deserial
root           2  0.0  0.0      0     0 ?        S    00:58   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   00:58   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   00:58   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   00:58   0:00 [kworker/0:0H-
xfs-log/nvme0n1p2]
root           8  0.0  0.0      0     0 ?        I<   00:58   0:00 [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    00:58   0:00 [ksoftirqd/0]
...
```

VSZ	Virtual memory SiZe，虚拟内存集，线性内存
RSS	ReSident Size, 常驻内存集
TTY	该进程所依附的终端
STAT	进程状态
START	进程启动的时间(HH:MM)或日期
TIME	累计的CPU时间
%CPU	(cputime/realtime)
%MEM	该进程的驻留内存[^1]相对于物理总内存的比例

- STAT列所代表的意思

R	running
S	interruptable sleeping
D	uninterruptable sleeping
T	stopped
Z	zombie
+	前台进程
l	多线程进程

L	内存分页并带锁
N	低优先级进程
<	高优先级进程
s	session leader，会话（子进程）发起者

- 一些ps命令实例

```
ps axo pid,cmd,psr,ni,pri,rtprio              # BSD风格选项使用o选项后紧跟自定义的显示项目列表
常用组合：
    aux
    -ef
    -eFH
    -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,comm
    axo stat,euid,ruid,tty,tpgid,sess,pgrp,ppid,pid,pcpu,comm

查询你拥有的所有进程
    ps -x
显示指定用户名(RUID)或用户ID的进程
    ps -fU apache
    ps -fU 48
显示指定用户名(EUID)或用户ID的进程
    ps -fu wang
    ps -fu 1000
查看以root用户权限（实际和有效ID）运行的每个进程
    ps -U root -u root
列出某个组拥有的所有进程（实际组ID：RGID或名称）
    ps -fG nginx
列出有效组名称（或会话）所拥有的所有进程
    ps -fg mysql
    ps -fg 27
显示指定的进程ID对应的进程
    ps -fp 1234
以父进程ID来显示其下所有的进程，如显示父进程为1234的所有进程
    ps -f --ppid 1234
显示指定PID的多个进程
    ps -fp 1204,1239,1263
要按tty显示所属进程
    ps -ft pts/0
以进程树显示系统中的进程如何相互链接
    ps -e --forest
以进程树显示指定的进程
    ps -f --forest -C sshd
    ps -ef --forest | grep -v grep | grep sshd
要显示一个进程的所有线程,将显示LWP（轻量级进程）以及NLWP（轻量级进程数）列
    ps -fL -C nginx
要列出所有格式说明符
    ps L
查看进程的PID，PPID，用户名和命令
```

```
    ps -eo pid,ppid,user,cmd
自定义格式显示文件系统组,ni值开始时间和进程的时间
    ps -p 1234 -o pid,ppid,fgroup,ni,lstart,etime
使用其PID查找进程名称：
    ps -p 1244 -o comm=
要以其名称选择特定进程，显示其所有子进程
    jps -C sshd,bash
查找指定进程名所有的所属PID，在编写需要从std输出或文件读取PID的脚本时这个参数很有用
    ps -C httpd,sshd -o pid=
检查一个进程的执行时间
    ps -eo comm,etime,user | grep nginx
查找占用最多内存和CPU的进程
    ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
    ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head
显示安全信息
    ps -eM
    ps --context
使用以下命令以用户定义的格式显示安全信息
    ps -eo euser,ruser,suser,fuser,f,comm,label
使用watch实用程序执行重复的输出以实现对就程进行实时的监视，如下面的命令显示每秒钟的监视
    watch -n 1 'ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head'
```

3.nice命令

- 使用nice命令指定一个调度优先级来运行某程序

```
nice [OPTION] [COMMAND [ARG]...]
[root@centos8 ~]#nice -n 5 sleep 333&
[root@centos8 ~]#ps axo pri,cmd,pid,ni
...
 19 [kworker/1:3-xfs-cil/nvme0n  64292   0
 19 [kworker/u256:2-events_unbo  64294   0
 14 sleep 333                    64305   5
 19 ps axo pri,cmd,pid,ni        64306   0
...

[root@centos8 ~]#renice -n 7 64305
64305 (process ID) old priority 5, new priority 7
...
 19 [kworker/1:3-cgroup_destroy  64292   0
 19 [kworker/u256:2-events_unbo  64294   0
 12 sleep 333                    64305   7
 19 [kworker/3:1+events]         64310   0
 19 [kworker/2:1-cgroup_pidlist  64322   0
...
```

4.renice命令

- renice命令可以更改一个正在运行的进程的优先级

```
renice [-n] priority pid...
```

- 查看

```
ps axo pid,comm,ni
```

```
[root@centos8 ~]#renice -n 7 64305
64305 (process ID) old priority 5, new priority 7
...
 19 [kworker/1:3-cgroup_destroy  64292   0
 19 [kworker/u256:2-events_unbo  64294   0
 12 sleep 333                    64305   7
 19 [kworker/3:1+events]         64310   0
 19 [kworker/2:1-cgroup_pidlist  64322   0
...
```

5.pgrep命令

- pgrep和pkill命令大部分选项相同，也就是大部分功能相同；但是pgrep一般用来基于进程名搜索某进程， pkill一般基于进程名来发送相关信号给某进程。
- 一般最灵活的搜索特定进程的方法为：ps 选项 | 其它命令
- 而pgrep则可以按预定义的模式搜索进程
- 用法

```
pgrep [options] pattern
-u uid: effective user，生效者
-U uid: real user，真正发起运行命令者
-t terminal: 与指定终端相关的进程
-l: 显示进程名
-a: 显示完整格式的进程名
-P pid: 显示指定进程的子进程
按确切的程序名称：/sbin/pidof
    pidof bash
```

6.kill命令

- kill命令一般用来结束某进程，但是其还有其他功能，用于发送特定的信号给相关进程来控制某进程。 以实现对进程管理,每个信号对应一个数字，信号名称以SIG开头（可省略），不区分大小写

```
显示当前系统可用信号：
    kill -l
    trap -l
常用信号：man 7 signal
1) SIGHUP   无须关闭进程而让其重读配置文件
2) SIGINT   中止正在运行的进程；相当于Ctrl+c
3) SIGQUIT  相当于ctrl+\
9) SIGKILL  强制杀死正在运行的进程
15) SIGTERM 终止正在运行的进程
18) SIGCONT 继续运行
19) SIGSTOP 后台休眠

指定信号的方法 ：
   (1) 信号的数字标识：1, 2, 9
   (2) 信号完整名称：SIGHUP
   (3) 信号的简写名称：HUP

按PID来给某个进行发送信号：kill [-SIGNAL]  pid …
    kill -n SIGNAL pid
    kill -s SIGNAL pid
按名称：killall [-SIGNAL]  comm…
按模式：pkill [options] pattern
    -SIGNAL
    -u uid: effective user，生效者
    -U uid: real user，真正发起运行命令者
    -t terminal: 与指定终端相关的进程
    -l: 显示进程名（pgrep可用）
    -a: 显示完整格式的进程名（pgrep可用）
    -P pid: 显示指定进程的子进程
```

```
 Signal     Value     Action   Comment
       ──────────────────────────────────────────────────────────────────────
       SIGHUP        1       Term    Hangup detected on controlling terminal
                                     or death of controlling process
       SIGINT        2       Term    Interrupt from keyboard

       SIGQUIT       3       Core    Quit from keyboard
       SIGILL        4       Core    Illegal Instruction
       SIGABRT       6       Core    Abort signal from abort(3)
       SIGFPE        8       Core    Floating-point exception
       SIGKILL       9       Term    Kill signal
       SIGSEGV      11       Core    Invalid memory reference
       SIGPIPE      13       Term    Broken pipe: write to pipe with no
                                     readers; see pipe(7)
       SIGALRM      14       Term    Timer signal from alarm(2)
       SIGTERM      15       Term    Termination signal
       SIGUSR1   30,10,16    Term    User-defined signal 1
       SIGUSR2   31,12,17    Term    User-defined signal 2
       SIGCHLD   20,17,18    Ign     Child stopped or terminated
       SIGCONT   19,18,25    Cont    Continue if stopped
       SIGSTOP   17,19,23    Stop    Stop process
```

```
SIGTSTP   18,20,24    Stop    Stop typed at terminal
SIGTTIN   21,21,26    Stop    Terminal input for background process
SIGTTOU   22,22,27    Stop    Terminal output for background process
```

- pgrep和pkill命令大部分选项相同，也就是大部分功能相同；但是pgrep一般用来基于进程名搜索某进程， pkill一般基于进程名(支持正则表达式模式匹配)来发送相关信号给某进程。
- killall命令单纯的基于进程名来结束某进程

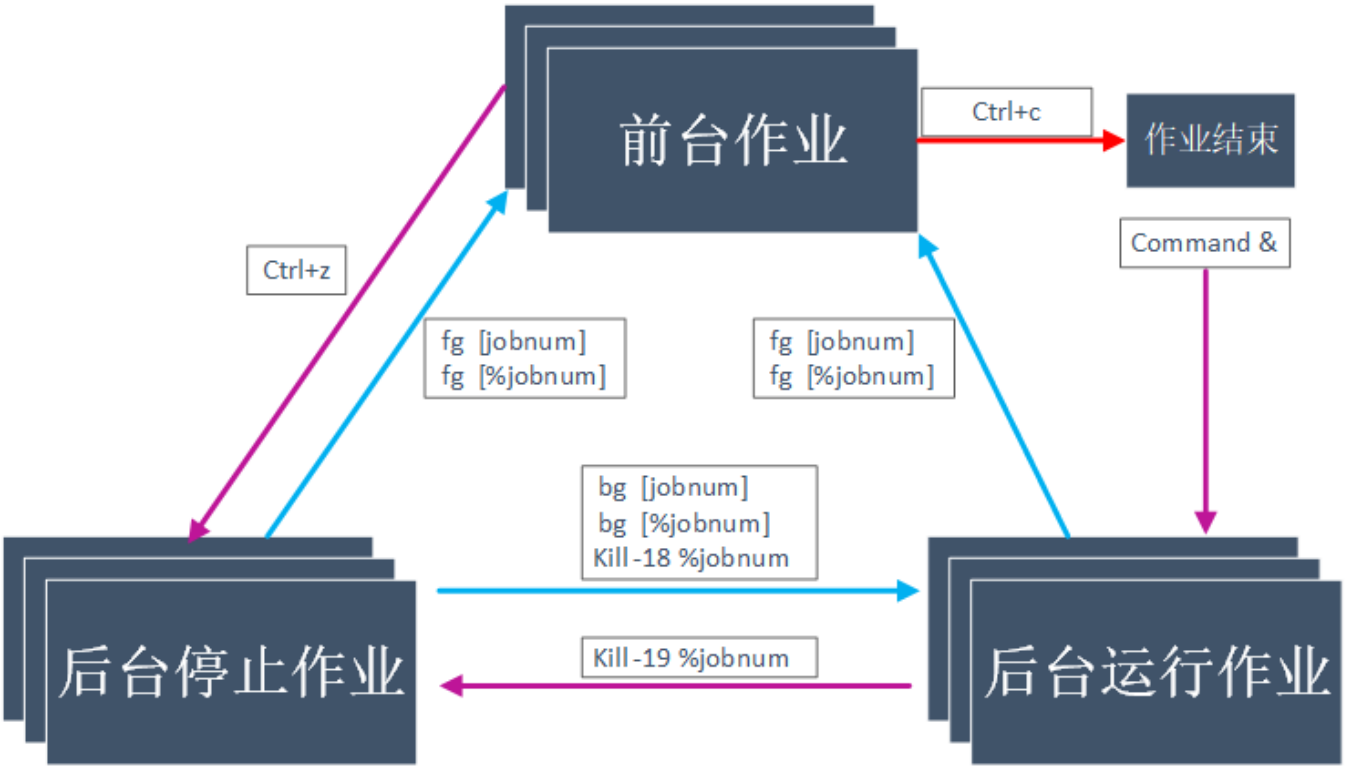
三.Linux系统计划任务和用户计划任务

1.什么是作业?

- 在Linux中的作业是指正在运行的还没有结束的命令或者任务。Linux是个支持多任务的操作系统， 其允许多个命令同时执行(多核CPU真正的同时执行；单核CPU并发执行)。每个正在进行的作业被唯 一的作业号所标识。
- 要查看和控制作业主要使用以下命令
    - jobs 列出正在运行或者挂起的作业
    - fg 将作业转换为前台作业
    - bg 将作业转换为后台作业
    - stop 挂起作业(Ctrl + z)
    - kill 结束作业(Ctrl + c)

2.Linux的前后台作业管理

- Linux中某个命令或脚本默认运行时为前台进程，可以使用Ctrl + c结束其；如果发起该命令或者 脚本的终端被关闭，那么该进程也就结束；这不利于需要长时间做某事的作业运行，在运行某命令或则 脚本时可以在其后加上&符号来使其以后台作业的方式运行，这样即使使用Ctrl+c或者关闭终端也不会 导致作业被终止。
- 作业状态切换图示：



- 例子：

```
[root@centos8 ~]#sleep 100 &
[1] 51898
[root@centos8 ~]#jobs
[1]+  Running                 sleep 100 &
```

- 上面的"1"为作业号(作业被当前的shell所维护)。
- "51898"是进程ID号即PID(进程由系统维护)
- 使用"kill %1"或者"kill 51898"结束该作业
- 使用bash内置命令disown来移除活动的作业
- 用法：disown PID

```
[root@centos8 ~]#sleep 2000 &
[1] 52837
[root@centos8 ~]#sleep 200 &
[2] 52838
[root@centos8 ~]#jobs
[1]-  Running                 sleep 2000 &
[2]+  Running                 sleep 200 &
[root@centos8 ~]#disown 52838                   # 指定作业PID来结束作业
[root@centos8 ~]#jobs
[1]+  Running                 sleep 2000 &
[root@centos8 ~]#disown sleep
[root@centos8 ~]#jobs
...
```

```
[root@centos8 ~]#sleep 200&
[1] 52855
[root@centos8 ~]#jobs
[1]+  Running                 sleep 200 &
[root@centos8 ~]#disown %1                      # 指定作业号来结束作业
[root@centos8 ~]#jobs
...
[root@centos8 ~]#sleep 200&
[1] 52868
[root@centos8 ~]#disown                         # 不指定任何信息，默认当前作业
[root@centos8 ~]#jobs
[root@centos8 ~]#
```

jobs

```
jobs -l    # 增加显示PID
jobs -n    # 当作业最后一次通知用户后状态有所改变时就显示相关信息
jobs -p    # 只显示作业进程号PID
jobs -r    # 只显示运行的作业
jobs -s    # 只显示停止的作业
```

fg

- fg恢复某个停止的后台作业或转换正在运行的后台作业为前台作业，并使之为当前作业。

```
[root@centos8 ~]#jobs
[1]-  Stopped                 sleep 2222
[2]   Running                 sleep 222546 &
[3]   Running                 sleep 222322 &
[4]+  Stopped                 sleep 2000
[5]   Running                 sleep 23424324 &
[root@centos8 ~]#fg %4
sleep 2000
^C
[root@centos8 ~]#
```

bg

- bg命令恢复某个挂起的后台作业为运行的后台作业

```
[root@centos8 ~]#jobs
[1]+  Stopped                 sleep 2222
[2]   Running                 sleep 222546 &
[3]   Running                 sleep 222322 &
[5]-  Running                 sleep 23424324 &
[root@centos8 ~]#bg %1
[1]+ sleep 2222 &
```

```
[root@centos8 ~]#jobs
[1]   Running                 sleep 2222 &
[2]   Running                 sleep 222546 &
[3]-  Running                 sleep 222322 &
[5]+  Running                 sleep 23424324 &
[root@centos8 ~]#
```

3.Linux中同时运行多个进程的方法(多核CPU)

- 方法一:在命令或者脚本名后加&符号使其后台运行

```
[root@centos8 ~]#sleep 20&
[1] 43980
```

```
#!/bin/bash
p1.sh&
p2.sh&
p3.sh&
```

- 方法二:使用圆括号来产生多个子进程，各子进程可以同时被运行

```
(job1&);(job2&);(job3&)
eg:
(p1.sh&);(p2sh&);(p3.sh&)
```

- 方法三:使用花括号

```
{p1.sh& p2sh& p3.sh&}
```

4.Linux任务计划

4.1未来某时间点执行一次的任务

- 使用at命令来定义未来某时间点执行一次的任务
- at命令所在包：at
- at 命令用法：

```
at [option] TIME
常用选项：
    -V 显示版本信息
```

```
    -t time 时间格式  [[CC]YY]MMDDhhmm[.ss]
    -l 列出指定队列中等待运行的作业；相当于atq
    -d 删除指定的作业；相当于atrm
    -c 查看具体作业任务
    -f /path/file 指定的文件中读取任务
    -m 当任务被完成之后，将给用户发送邮件，即使没有标准输出
注意：作业执行命令的结果中的标准输出和错误以邮件通知给相关用户

TIME：定义出什么时候进行 at 这项任务的时间
    HH:MM [YYYY-mm-dd]
    noon, midnight, teatime（4pm）
    tomorrow
    now+#{minutes,hours,days, OR weeks}

HH:MM  02:00
    在今日的 HH:MM 进行，若该时刻已过，则明天此时执行任务
HH:MM YYYY-MM-DD   02:00 2016-09-20
    规定在某年某月的某一天的特殊时刻进行该项任务
HH:MM[am|pm] [Month] [Date]
    04pm March 17
    17:20 tomorrow
HH:MM[am|pm] + number [minutes|hours|days|weeks]
    在某个时间点再加几个时间后才进行该项任务
        now + 5 min
        02pm + 3 days
```

- at任务的执行方式

```
执行方式：
    1）交互式
    2）输入重定向
    3）at -f 文件
依赖于atd服务,需要启动才能实现at任务
at队列存放在/var/spool/at目录中
/etc/at.{allow,deny}文件控制用户是否能执行at任务,如果不存在则手工创建
    白名单：/etc/at.allow 默认不存在，只有该文件中的用户才能执行at命令
    黑名单：/etc/at.deny 默认存在，拒绝该文件中用户执行at命令，而没有在
at.deny 文件中的使用者则可执行
    如果两个文件都不存在，只有 root 可以执行 at 命令
```

- 例子

```
[root@centos7 ~]#at 16:30
at> echo hahah
at> <EOT>                                  # Ctrl+d 退出作业编辑
job 3 at Sat Oct 26 16:30:00 2019

[root@centos7 ~]#at -l                     # 列出作业队列
1       Sun Oct 27 16:15:00 2019 a root
```

```
2       Sun Oct 27 16:24:00 2019 a root
3       Sat Oct 26 16:30:00 2019 a root

[root@centos7 ~]#tree /var/spool/at
/var/spool/at
|-- a00001018fd26f
|-- a00002018fd278
|-- a00003018fccde                    # 当前计划作业
|-- spool

[root@centos7 ~]#cat /var/spool/at/a00003018fccde
#!/bin/sh
# atrun uid=0 gid=0
# mail root 0
umask 22
XDG_SESSION_ID=75; export XDG_SESSION_ID
HOSTNAME=centos7.magedu.steve; export HOSTNAME
SHELL=/bin/bash; export SHELL
...
...
${SHELL:-/bin/sh} <\< 'marcinDELIMITER4bbe8c69'
echo hahah
marcinDELIMITER4bbe8c69

[root@centos7 ~]#mail                  # 作业完成后发邮件通知用户
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N  1 root                  Sat Oct 26 16:30  14/494   "Output from your jo"
& 1
Message  1:
From root@centos7.magedu.steve  Sat Oct 26 16:30:00 2019
Return-Path: <root@centos7.magedu.steve>
X-Original-To: root
Delivered-To: root@centos7.magedu.steve
Subject: Output from your job        3
To: root@centos7.magedu.steve
Date: Sat, 26 Oct 2019 16:30:00 +0800 (CST)
From: root@centos7.magedu.steve (root)
Status: R

hahah                                # 作业的标准输出内容

&
```

4.2未来周期性执行的任务

4.2.1实现cron计划任务的前提

- 安装相关程序包

```
cronie
    主程序包，提供crond守护进程及相关辅助工具
crontabs
    包含CentOS提供系统维护任务
cronie-anacron
    cronie的补充程序，用于监控cronie任务执行状况，如cronie中的任务在过去应该运行的时间
    点未能正常运行，则anacron会随后启动一次此任务
```

- 确保crond守护处于运行状态

```
CentOS 7:
    systemctl status crond
CentOS 6:
    service crond status
```

- 计划周期性执行的任务提交给crond，到指定时间会自动运行
- 系统cron任务：系统维护的周期性作业,位于： /etc/crontab文件内，一般只有root可以更改。
- 用户cron任务：使用crontab命令来创建用户各自的周期性任务
- 日志：/var/log/cron
- 使用crontab命令来定义周期性计划任务

4.2.2系统cron计划任务

- 系统cron任务:/etc/crontab
- 注释行以 # 开头
- 详情参见 man 4 rontab

```
[root@centos7 ~]#cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
```

- 例如：晚上9点10分以用户steve的身份运行echo命令

> 10 21 * * * steve /bin/echo "Hello there!!"

- 时间表示的方法

```
时间表示法：
(1) 特定值
    给定时间点有效取值范围内的值
(2) *
    给定时间点上有效取值范围内的所有值
    表示“每...”
(3) 离散取值
    #,#,#
(4) 连续取值
    #-#
(5) 在指定时间范围上，定义步长
    /#: #即为步长
```

- 可以替换的时间格式

```
@yearly   0 0 1 1 *
@annually 0 0 1 1 *
@monthly  0 0 1 * *
@weekly   0 0 * * 0
@daily    0 0 * * *
@hourly   0 * * * *
@reboot   Run once after reboot
示例：每3小时echo和wall命令
0 */3 * * * wang /bin/echo “howdy”;  wall “welcome to Magedu!”
```

- 例子:每3小时echo和wall命令

> 0 */3 * * * steve /bin/echo "Done something!"; wall "Done something again!"

- 系统计划任务所涉及的配置文件及任务队列

```
配置文件 /etc/crontab
配置文件 /etc/cron.d/
脚本 /etc/cron.hourly/
脚本 /etc/cron.daily/
脚本 /etc/cron.weekly/
脚本 /etc/cron.monthly/
```

4.2.3anacron系统

```
运行计算机关机时cron不运行的任务，CentOS6以后版本取消anacron服务，由crond服务管理
对笔记本电脑、台式机、工作站、偶尔要关机的服务器及其它不一直开机的系统很重要对很有用
配置文件：/etc/anacrontab，负责执行/etc/ cron.daily /etc/cron.weekly
/etc/cron.monthly中系统任务
    •字段1：如果在这些日子里没有运行这些任务……
    •字段2：在重新引导后等待这么多分钟后运行它
    •字段3：任务识别器，在日志文件中标识
    •字段4：要执行的任务
由/etc/cron.hourly/0anacron执行
当执行任务时，更新/var/spool/anacron/cron.daily 文件的时间戳
```

4.2.4管理临时文件

```
CentOS7使用systemd-tmpfiles-setup服务实现
CentOS6使用/etc/cron.daily/tmpwatch定时清除临时文件
配置文件：
    /etc/tmpfiles.d/*.conf
    /run/tmpfiles.d/*.conf
    /usr/lib/tmpfiles/*.conf
/usr/lib/tmpfiles.d/tmp.conf
    d /tmp 1777 root root 10d
    d /var/tmp 1777 root root 30d
命令：
    systemd-tmpfiles –clean|remove|create  configfile
```

4.2.5用户计划任务

- crontab命令定义
- 每个用户都有专用的cron任务文件：/var/spool/cron/USERNAME
- crontab命令：

```
crontab [-u user] [-l | -r | -e] [-i]
-l 列出所有任务
-e 编辑任务
-r 移除所有任务
-i 同-r一同使用，以交互式模式移除指定任务
-u user 仅root可运行，指定用户管理cron任务
```

- 控制用户执行计划任务：/etc/cron.{allow,deny}

4.2.6 at和crontab对比

- 一次性作业使用 at
- 重复性作业使用crontab

Create	at TIME	crontab -e
List	at -l	crontab -l
Details	at -c jobnum	crontab -l
Remove	at -d jobnum	crontab -r
Edit	N/A	crontab -e

- 没有被重定向的输出会被邮寄给用户
- root能够修改其它用户的作业
- 注意：运行结果的标准输出和错误都以邮件通知给相关用户

```
(1) COMMAND > /dev/null
(2) COMMAND &> /dev/null
```

- 对于cron任务来讲，%有特殊用途；如果在命令中要使用%，则需要转义，将% 放置于单引号中，则可不用转义

思考

- (1) 如何在秒级别运行任务？

```
* * * * * for min in 0 1 2; do echo "hi"; sleep 20; done
```

- (2) 如何实现每7分钟运行一次任务?

```
*/6 * * * * for min in 0 1 2; do sleep 1m; echo "hi"; done
```

```
sleep命令：
    sleep NUMBER[SUFFIX]...
SUFFIX:
    s: 秒，默认
    m: 分
    h: 小时
    d: 天
```

5.练习

1、每周的工作日1：30，将/etc备份至/backup目录中，保存的文件名称格式为"etcbak-yyyy-mm-dd-HH.tar.xz"，其中日期是前一天的时间

```
vim backup.sh
#!/bin/bash
tar Jcf /backup/etcbak-`date +%F-%H`.tar.xz /etc/ &
...
chmod +x backup.sh
crontab -e
30 1 * * 1-5 /root/backup.sh
```

2、每两小时取出当前系统/proc/meminfo文件中以S或M开头的信息追加至/tmp/meminfo.txt文件中

```
vim meminfo.sh
#!/bin/bash
INFO=`grep -E '^(S|M).*' /proc/meminfo`
touch /proc/meminfo
cat "$INFO" >> /proc/meminfo &> /dev/null
...
chmod +x meminfo.sh
crontab -e
0 */2 * * * /root/meminfo.sh
```

3、工作日时间，每10分钟执行一次磁盘空间检查，一旦发现任何分区利用率高于80%，就执行wall警报

```
vim check.sh
#!/bin/bash
THRESHOLD=80
D_MAX_USAGE=`df  | sed -nr '/^\/dev\/sd.*/s#.* (.+)% .*#\1#p' | sort -nr | he ad -
n1`
I_MAX_USAGE=`df -i  | sed -nr '/^\/dev\/sd.*/s#.* (.+)% .*#\1#p' | sort -nr | he
ad -n1`
if [[ "$THRESHOLD"  -eq "$D_MAX_USAGE"]]; do
    wall "Disk usage almost full!!"
elif [[ "$THRESHOLD"  -eq "$I_MAX_USAGE"]]; do
    wall "Inode usage almost full!!"
done
...
chmod +x check.sh
crontab -e
*/10 * * * 1-5 /root/check.sh
```

注脚

[^1]:驻留内存：即RSS(resident set size)是驻留集合大小，即进程所使用的非交换区的物理内存。