day05-进程管理-supervisor-计划任务-systemd

5.1进程管理

5.1.1 什么是进程

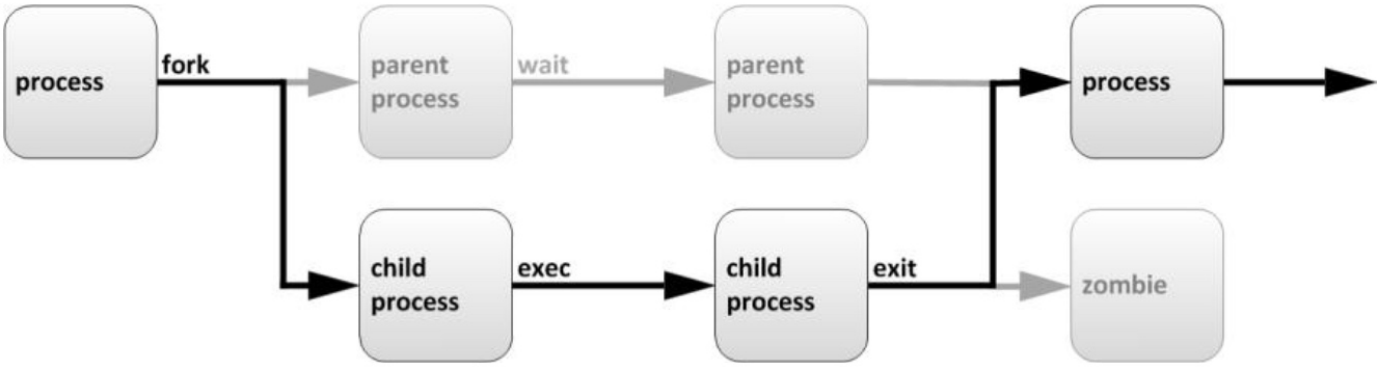
- 开发写的代码我们称为程序，那么将开发的代码运行起来。我们称为进程。
- 总结一句话就是：当我们运行一个程序，那么我们将运行的程序叫进程。
- `PS1`：当程序运行为进程后，系统会为该进程分配内存，以及进程运行的身份和权限。
- `PS2`：在进程运行的过程中，系统会有各种指标来表示当前运行的状态。

5.1.2 程序和进程的区别

- 1.程序是数据和指令的集合，是一个静态的概念。
    - 比如 `/bin/ls、/bin/cp` 等二进制文件。同时程序可以长期存在系统中。
- 2.进程是程序运行的过程，是一个动态的概念。
    - 进程是存在生命周期的概念的，也就是说进程会随着程序的终止而销毁，不会永久存在系统中。

5.1.3 进程的生命周期

- 生命周期就是指一个对象的生老病死。用处很广



当父进程接收到任务调度时，会通过fock派生子进程来处理，那么子进程会继承父进程属性。
1.子进程在处理任务代码时，父进程其实不会进入等待，其运行过程是由linux系统进行调度的。
2.子进程在处理任务代码后，会执行退出，然后唤醒父进程来回收子进程的资源。
3.如果子进程在处理任务代码过程中异常退出，而父进程却没有回收子进程资源，会导致子进程虽然运行实体已经消失，但仍然在内核的进程表中占据一条记录，长期下去对于系统资源是一个浪费。（僵尸进程）
4.如果子进程在处理任务过程中，父进程退出了，子进程没有退出，那么这些子进程就没有父进程来管理了，由系统的system进程管理。（孤儿进程）
PS: 每个进程都父进程的PPID，子进程则叫PID。
例：假设现在我是蒋先生(system进程)....故事持续中.....僵尸进程与孤儿进程区别

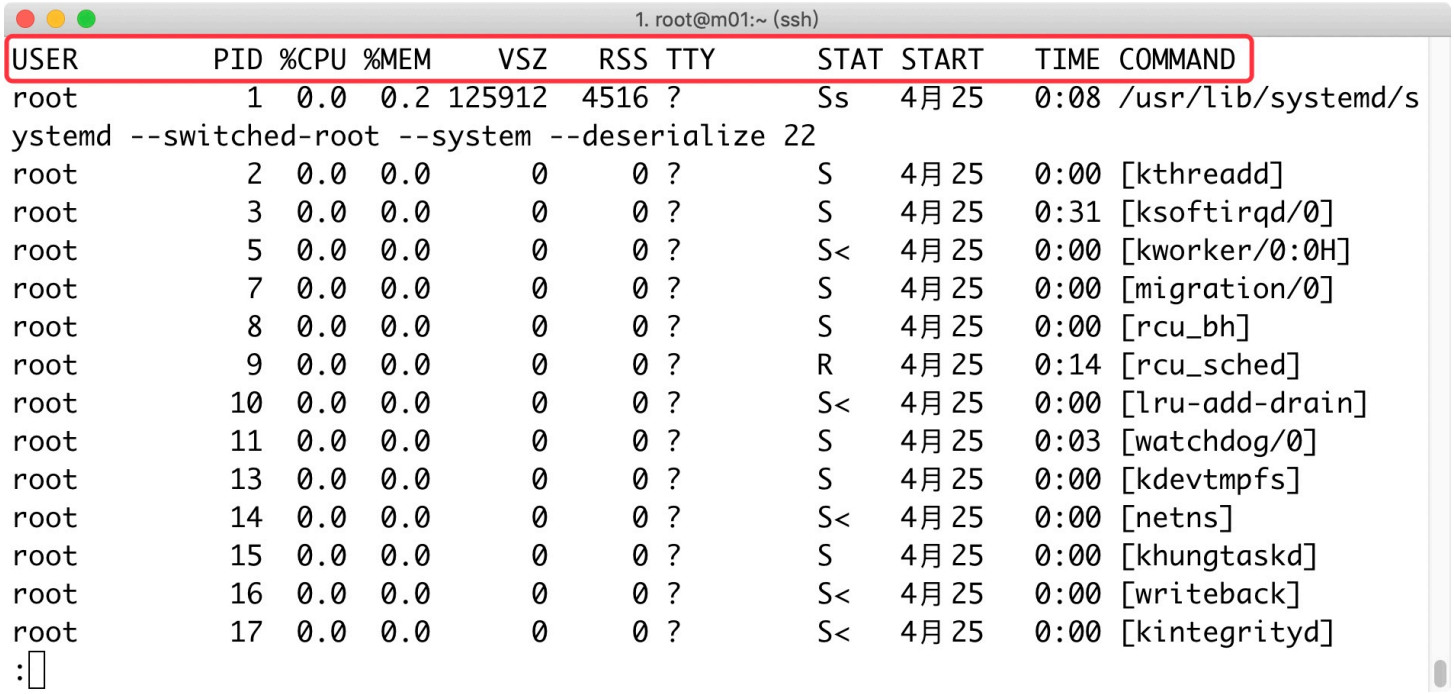
5.1.4 监控进程状态

- 程序在运行后，我们需要了解进程的运行状态。查看进程的状态分为：
  - 静态查看
  - 动态查看

5.1.4.1 静态查看进程ps

- `ps -aux` 常用组合，查看进程 用户、PID、占用cpu百分比、占用内存百分比、状态、执行的命令等

5.4.1.1.1 每列含义详解

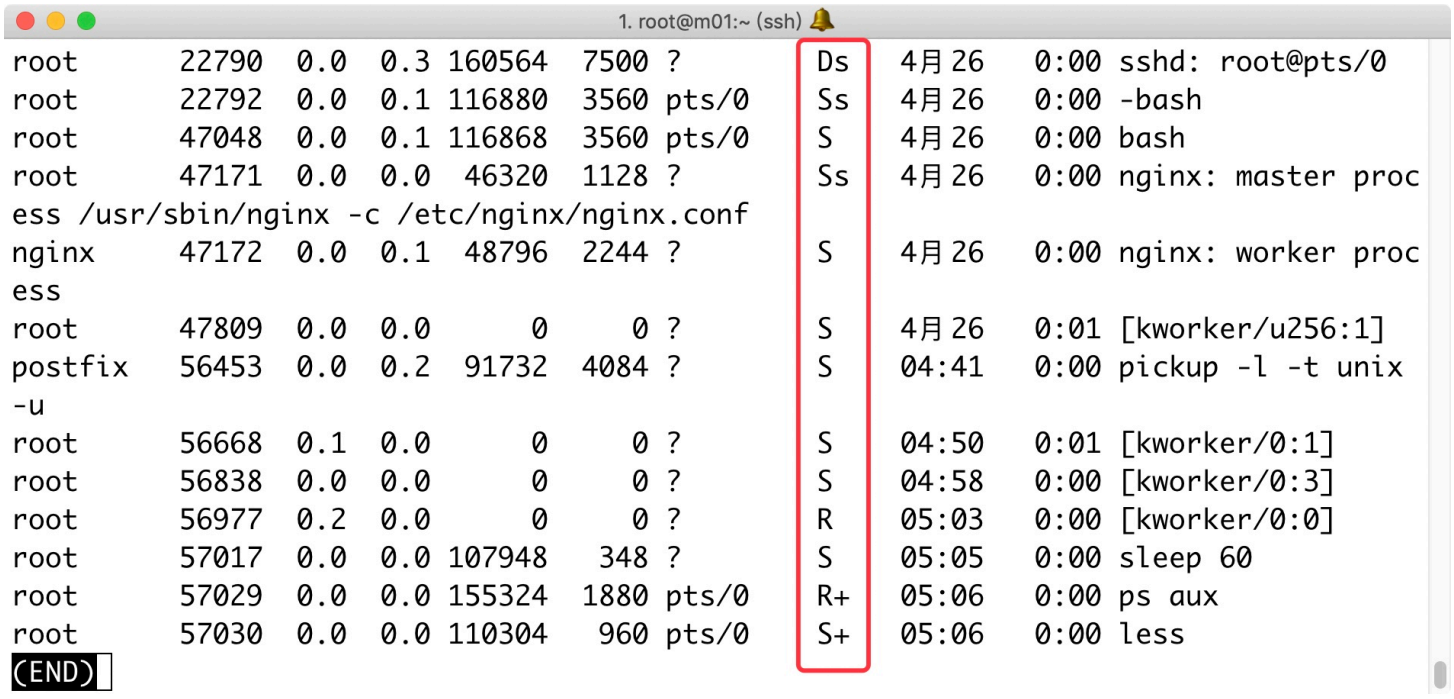


状态	描述
USER	启动进程的用户

PID	进程运行的ID号
%CPU	进程占用CPU百分比
%MEM	进程占用内存百分比
VSZ	进程占用虚拟内存大小 (单位KB)
RSS	进程占用物理内存实际大小 (单位KB)
TTY	进程是由哪个终端运行启动的tty1、pts/0等 ?表示内核程序与终端无关
STAT	进程运行过程中的状态 man ps （/STATE）
START	进程的启动时间
TIME	进程占用 CPU 的总时间(为0表示还没超过秒)
COMMAND	程序的运行指令，[方括号] 属于内核态的进程。 没有 [ ] 的是用户态进程。

5.1.4.1.2 STAT状态含义

- `STAT` 状态的 `S、Ss、S+、R、R、S+` 等等，都是什么意思?



STAT基本状态	描述	STAT状态+符号	描述
R	进程运行	s	进程是控制进程， Ss进程的领导者，父进程
	可中断睡		进程运行在高优先级上，S<优先级较

S	眠	<	高的进程
T	进程被暂停	N	进程运行在低优先级上，SN优先级较低的进程
D	不可中断进程	+	当前进程运行在前台，R+该表示进程在前台运行
Z	僵尸进程	l	进程是多线程的，Sl表示进程是以线程方式运行

5.1.4.1.3 进程状态切换-案例1

1.在终端1上运行 `vim`

```
[root@xuliangwei ~]# vim new_file
```

2.在终端2上运行 `ps` 命令查看状态

```
[root@xuliangwei ~]# ps aux|grep new_file    # S表示睡眠模式，+表示前台运行
root       58118  0.4  0.2 151788  5320 pts/1    S+   22:11   0:00 new_file
root       58120  0.0  0.0 112720   996 pts/0    R+   22:12   0:00 grep --color=auto
 new_file

# 在终端1上挂起vim命令，按下：ctrl+z
```

3.回到终端2再次运行 `ps` 命令查看状态

```
[root@xuliangwei ~]# ps aux|grep new_file    # T表示停止状态
root       58118  0.1  0.2 151788  5320 pts/1    T    22:11   0:00 vim new_file
```

5.1.4.1.4 不可中断进程-案例2

- 使用 `tar` 打包文件时，可以通过终端不断查看状态，由`S+，R+、D+`

```
[root@xuliangwei ~]# tar -czf etc.tar.gz /etc/ /usr/ /var/

[root@xuliangwei ~]# ps aux|grep tar|grep -v grep
root       58467  5.5  0.2 127924  5456 pts/1    R+   22:22   0:04 tar -czf etc.tar.
gz /etc/
[root@xuliangwei ~]# ps aux|grep tar|grep -v grep
```

```
root      58467  5.5  0.2 127088  4708 pts/1    S+   22:22   0:03 tar -czf etc.tar.
gz /etc/
[root@xuliangwei ~]# ps aux|grep tar|grep -v grep
root      58467  5.6  0.2 127232  4708 pts/1    D+   22:22   0:03 tar -czf etc.tar.
gz /etc/
```

5.1.4.1.5 僵尸进程模拟-案例3

1.编写c语言程序，然后对该程序进行编译；

```
[root@dns-master ~]# cat z.c
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
pid_t pid;
pid = fork();
if (pid == 0) {
    int iPid = (int)getpid();
    fprintf(stderr,"我是子进程,%d\n",iPid);
    sleep(1);
    fprintf(stderr, "Child exits\n");
    return EXIT_SUCCESS;
}

int iPid = (int)getpid();
    fprintf(stderr,"我是父进程,%d\n",iPid);
    fprintf(stderr, "sleep....\n");
    sleep(360);
    fprintf(stderr, "parent exits\n");
    return EXIT_SUCCESS;
}
```

2.编译c语言程序，然后启动进程；

```
[root@dns-master ~]# gcc z.c
[root@dns-master ~]# ./a.out
我是父进程,17781
sleep....
我是子进程,17782
```

```
Child exits
```

3.通过 `ps aux` 查看进程状态；

```
[root@dns-master ~]# ps aux |grep a.out
root      17781  0.0  0.0   4208   344 pts/0    S+   11:18   0:00 ./a.out
root      17782  0.0  0.0      0     0 pts/0    Z+   11:18   0:00 [a.out] <defunct>
```

4.找到僵死进程的父进程，kill掉父进程，那么僵死进程将变为孤儿进程，孤儿进程在系统中由init进程接管，init进程将回收僵死进程的资源；或者 `reboot` ，因为僵尸进程不会被杀死；

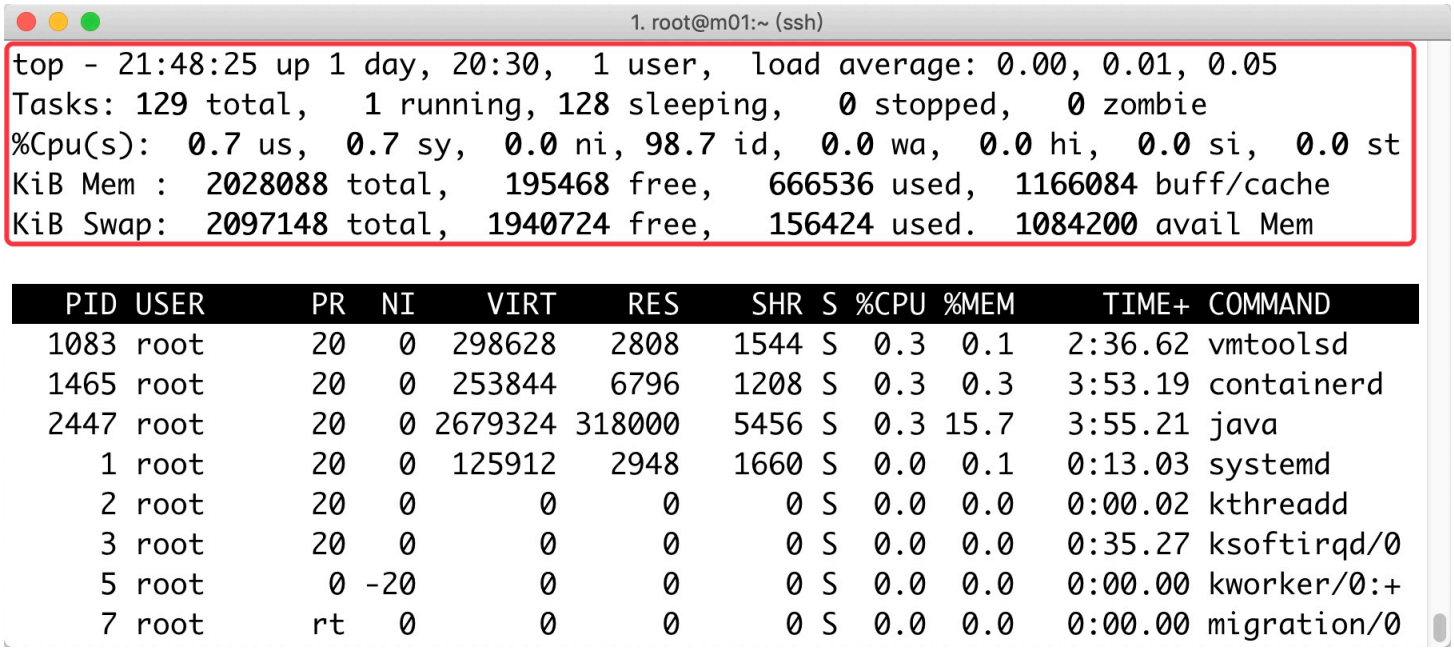
```
# kill 僵尸进程（子进程）
[root@dns-master ~]# kill 17782
[root@dns-master ~]# kill -9 17782
[root@dns-master ~]# ps aux |grep a.out
root      17781  0.0  0.0   4208   344 pts/0    S+   11:18   0:00 ./a.out
root      17782  0.0  0.0      0     0 pts/0    Z+   11:18   0:00 [a.out] <defunct>

# kill 僵尸进程（父进程）
[root@dns-master ~]# kill -9 17781
[root@dns-master ~]# ps aux |grep a.out
```

5.1.4.2 动态查看进程top

- 使用top命令查看当前的进程状态(动态)

字母	含义
h	查看帮出
1	数字1，显示所有CPU核心的负载
z	以高亮显示数据
b	高亮显示处于R状态的进程
M	按内存使用百分比排序输出
P	按CPU使用百分比排序输出
q	退出top

5.1.4.2.1 每列含义详解



任务	含义
Tasks: 129 total	当然进程的总数
1 running	正在运行的进程数
128 sleeping	睡眠的进程数
0 stopped	停止的进程数
0 zombie	僵尸进程数
%Cpu(s): 0.7 us	系统用户进程使用CPU百分比
0.7 sy	内核中的进程占用CPU百分比，通常内核是于硬件进行交互
98.7 id	空闲CPU的百分比
0.0 wa	CPU等待IO完成的时间
0.0 hi	硬中断，占的CPU百分比
0.0 si	软中断，占的CPU百分比
0.0 st	比如虚拟机占用物理CPU的时间

5.1.4.2.2 如何理解中断

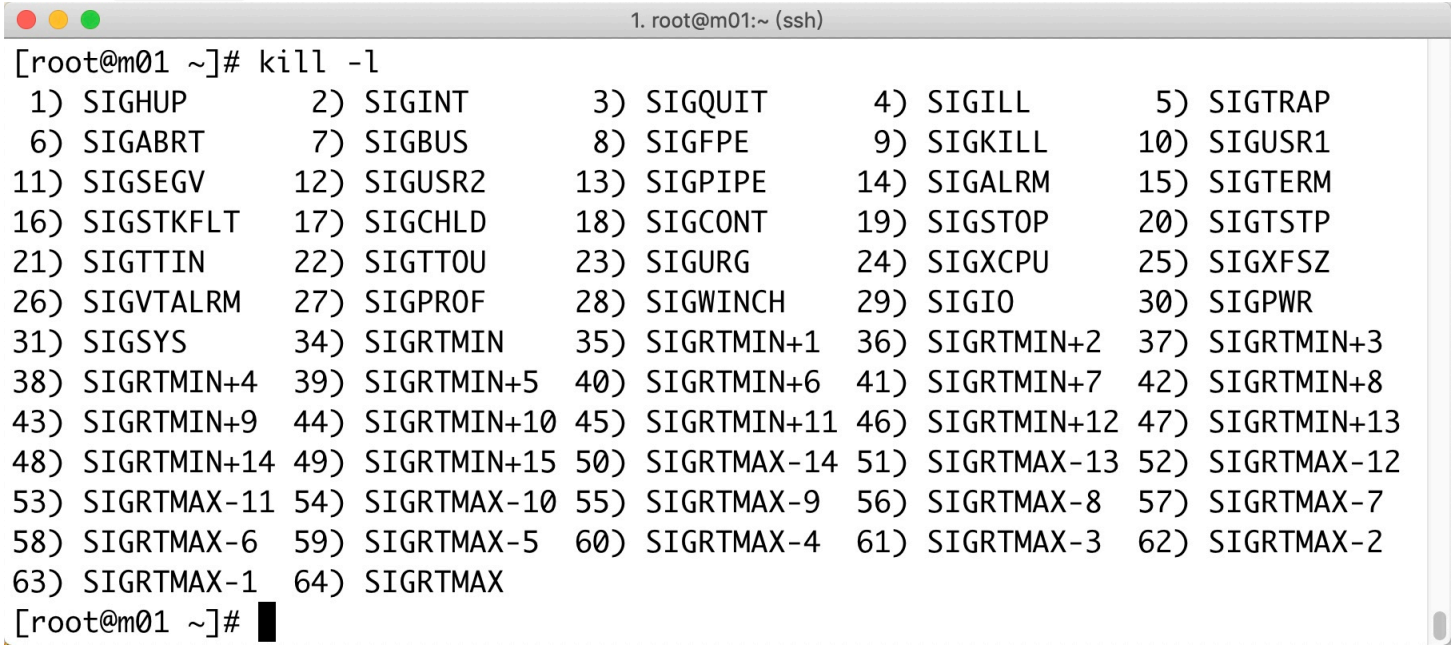
- 如何理解中断

5.1.5 管理进程状态

当程序运行为进程后，如果希望停止进程，怎么办呢？那么此时我们可以使用 `linux` 的 `kill` 命令对进程发送关闭信号。当然除了 `kill` 还有 `killall，pkill`

5.1.5.1 系统支持的信号

1.使用 `kill -l` 列出当前系统所支持的信号



2.虽然 `linux` 支持的信号很多，但是我们仅列出我们最为常用的3个信号

数字编号	信号含义	信号翻译
1	SIGHUP	通常用来重新加载配置文件
9	SIGKILL	强制杀死进程
15	SIGTERM	终止进程，默认kill使用该信号

5.1.5.2 关闭进程kill

- 使用 `kill` 命令，然后指定要杀死的进程 `PID`，即可停止该进程；

1.安装 `vsftpd` 服务，然后启动；

```
[root@xuliangwei ~]# yum -y install vsftpd
[root@xuliangwei ~]# systemctl start vsftpd
[root@xuliangwei ~]# ps aux|grep vsftpd
```

2.发送重载信号，例如 `vsftpd` 的配置文件发生改变，希望重新加载

```
[root@xuliangwei ~]# kill -1 9160
```

3.发送停止进程信号

```
[root@xuliangwei ~]# kill 9160
```

4.发送强制停止信号，当无法停止服务时，可强制终止信号；

```
[root@xuliangwei ~]# kill -9 9160
```

5.1.5.3 关闭进程pkill

- `Linux`系统中的 `killall、pkill` 命令主要用于杀死指定名字的进程。

示例1：通过 `pkill、killall` 指定服务名称，然后将其进程杀死

```
[root@xuliangwei ~]# pkill nginx
[root@xuliangwei ~]# killall nginx
```

示例2：使用 `pkill` 将远程连接用户 `t` 下线；

```
[root@xuliangwei ~]# pkill -9 -t pts/0
```

5.1.6 后台进程管理

5.1.6.1 什么是后台进程

通常进程都会在终端前台运行，一旦关闭终端，进程也会随着结束；
那么此时我们就希望进程能在后台运行，就是将在前台运行的进程放入后台运行，这样及时我们关闭了终端也不影响进程的正常运行。

5.1.6.2 为什么需要后台运行

比如：我们此前在国内服务器往国外服务器传输大文件时，由于网络的问题需要传输很久，如果在传输的过程中出现网络抖动或者不小心关闭了终端则会导致传输失败，如果能将传输的进程放入后台，是不是就能解决此类问题了。

5.1.6.3 如何将进程转为后台

早期的时候大家都选择使用 `nohup + &` 符号将进程放入后台，然后在使用 `jobs、bg、fg` 等方式查看进程状态，但太麻烦了 也不直观，所以我们推荐使用 `screen` 。

5.1.6.3.1 nohup方式

1.使用 `nohup` 将前台进程转换后台运行

```
[root@xuliangwei ~]# nohup sleep 3000 &
```

2.查看进程运行情况

```
[root@xuliangwei ~]# ps aux |grep sleep
root      75118  74766  0 11:10 pts/0    00:00:00 sleep 3000
```

3.使用 `job bg fg` 等方式查看后台作业

```
[root@xuliangwei ~]# jobs  # 查看后台作业
[1]+ Running sleep 3000 &

[root@ansible ~]# fg %1    # 转为前台运行
[root@ansible ~]# bg %1    # 转为后台运行
```

5.1.6.3.2 screen方式

1.安装 `screen` 工具

```
[root@web ~]# yum install screen -y
```

2.开启一个 `screen` 子窗口，可以通过 `-S` 为其指定名称

```
[root@web ~]# screen -S wget_soft
```

3.在 `screen` 窗口中执行任务，可以执行前台运行的任务

```

```

4.平滑退出 `screen` ，但不会终止 `screen` 中的前台任务

```
    # ctrl+a+d
```

5.查看当前有多少 `screen` 正在运行

```
[root@web ~]# screen -list
There is a screen on:
    22058.wget_soft (Detached)
1 Socket in /var/run/screen/S-root.
```

6.可以通过 `screenid` 或 `screen` 标签名称进入

```
[root@web ~]# screen -r wget_soft
[root@web ~]# screen -r 22058
[root@web ~]# exit                    # 退出进程，结束screen
```

5.1.7 进程的优先级

5.1.7.1 什么优先级

- 优先级指的是优先享受资源，比如排队买票时，军人优先、老人优先。等等*

5.1.7.2 为什么需要优先级

- 举个例子：海底捞火锅正常情况下响应就特别慢，那么当节假日时人员突增会导致处理请求特别慢；
  - 那假设我是海底捞VIP客户(最高优先级)，无论多么繁忙，我都无需排队，海底捞人员会直接服务于我，满足我的需求。
  - 如果我不是VIP的人员(较低优先级)则进入排队等待状态。(PS: 至于等多久，那.....)

5.1.7.3 如何为进程配置优先级

- 在启动进程时，为不同的进程使用不同的调度策略。
  - `nice` 值越高： 表示优先级越低，例如+19，该进程容易将CPU 使用量让给其他进程。
  - `nice` 值越低： 表示优先级越高，例如-20，该进程不倾向于让出CPU。*

5.1.7.3.1 如何查看进程优先级

1.使用 `top` 命令查看优先级；

```
# NI：显示nice值，默认是0。
```

```
# PR: 显示nice值, -20映射到0, +19映射到39
PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
1083 root      20   0  298628   2808   1544 S  0.3  0.1   2:49.28 vmtoolsd
5    root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:+
```

2.使用 `ps` 命令查看进程优先级；

```
[root@m01 ~]# ps axo command,nice |grep sshd|grep -v grep
/usr/sbin/sshd -D             0
sshd: root@pts/2              0
```

5.1.7.3.2 使用nice指定程序优先级

- `nice` 用于指定新启动程序的优先级。
  - 语法格式 `nice -n 优先级数字 进程名称`

1.启动 `vim` 并且指定程序优先级为 `-5`

```
[root@m01 ~]# nice -n -5 vim &
[1] 98417
```

2.查看当前 `vim` 进程的优先级

```
[root@m01 ~]# ps axo pid,command,nice |grep 98417
 98417 vim                        -5
```

5.1.7.3.3 使用renice修改进程优先级

- `renice` 命令用于修改一个正在运行的进程优先级。
  - 语法格式 `renice -n 优先级数字 进程pid`

1.查看当前正在运行的 `sshd` 进程优先级状态

```
[root@m01 ~]# ps axo pid,command,nice |grep [s]shd
 70840 sshd: root@pts/2              0
 98002 /usr/sbin/sshd -D             0
```

2.调整 `sshd` 主进程的优先级

```
[root@m01 ~]# renice -n -20 98002
98002 (process ID) old priority 0, new priority -20
```

3.调整之后需要退出终端，重新打开一个新终端

```
[root@m01 ~]# ps axo pid,command,nice |grep [s]shd
 70840 sshd: root@pts/2              0
 98002 /usr/sbin/sshd -D           -20
[root@m01 ~]# exit
```

4.再次登陆 `sshd` 服务，会由主进程 `fork` 子 `sshd` 进程(那么子进程会继承主进程的优先级)

```
[root@m01 ~]# ps axo pid,command,nice |grep [s]shd
 98002 /usr/sbin/sshd -D           -20
 98122 sshd: root@pts/0            -20
```

5.1.8.4 进程优先级案例

- 生产案例、Linux出现假死，怎么办，又如何通过nice解决?

5.1.8 系统平均负载

每次发现系统变慢时，我们通常做的第一件事，就是执行 top 或者 uptime 命令，来了解系统的负载情况。比如像下面这样，我在命令行里输入了 uptime 命令，系统也随即给出了结果。

```
[root@m01 ~]# uptime
 04:49:26 up 2 days,  2:33,  2 users,  load average: 0.70, 0.04, 0.05
#我们已经比较熟悉前面几列，它们分别是当前时间、系统运行时间以及正在登录用户数。

# 最后三个数字依次则是过去 1 分钟、5 分钟、15 分钟的平均负载（Load Average）。
```

5.1.8.1 什么是平均负载

- 平均负载是单位时间内的 CPU 使用率吗；
  - 上面的 0.70，就代表 CPU 使用率是 70%。其实不是；
- 那如何理解平均负载:
  - 平均负载是指单位时间内，系统处于"可运行状态"和"不可中断状态"的平均进程数，也就是平均活跃进程数；或者简单理解"平均负载"是"单位时间内的活跃进程数"；
  - 平均负载与 `CPU` 使用率并没有直接关系。

5.1.8.1.1 可运行状态

- 可运行状态进程：
- 指正在使用 `CPU` 或者正在等待 `CPU` 的进程，也就是我们 `ps` 命令看到处于 `R` 状态的进程

5.1.8.1.2 不可中断状态

- 不可中断进程：
- 系统中最常见的是等待硬件设备的 `I/O` 响应，通过 `ps` 命令中看到 `D（Disk Sleep）` 的进程。
- 例如：当一个进程向磁盘读写数据时，为了保证数据的一致性，在得到磁盘回复前，它是不能被其他进程或者中断打断的，这个时候的进程就处于不可中断状态。如果此时的进程被打断了，就容易出现磁盘数据与进程数据不一致的问题；
- 所以，不可中断状态实际上是系统对进程和硬件设备的一种保护机制；

5.1.8.2 平均负载合理设定

- 理想的状态是每个 `CPU` 上都刚好运行着一个进程，这样每个 `CPU` 都得到了充分利用。
- 所以在评判平均负载时，首先你要知道系统有几个 `CPU` 通过 `top` 命令获取，或 `/proc/cpuinfo`
- 示例：假设现在在 `4、2、1` 核的 `CPU` 上，如果平均负载为 `2` 时，意味着什么；
  - 1.在4个 `CPU` 的系统上，意味着 `CPU` 有 `50%` 的空闲；
  - 2.在2个 `CPU` 的系统上，意味着所有的 `CPU` 都刚好被完全占用；
  - 3.而1个 `CPU` 的系统上，则意味着有一半的进程竞争不到 `CPU`；

5.1.8.2.1 平均负载三大指标

- 实际上，平均负载中的三个指标我们其实都需要关注。（就好比一天的天气要结合起来看；）
- 1 分钟、5 分钟、15 分钟的三个值基本相同，或者相差不大，那就说明系统负载很平稳；
- 1分钟的值小于15分钟的值，说明系统最近1分钟的负载在减少，而过去 15 分钟内却有很大的负载；
- 如果 1 分钟的值远大于 15 分钟的值，就说明最近 1 分钟的负载在增加，这种增加有可能只是临时性的，也有可能还会持续上升，所以就需要持续观察。
- 一旦 1 分钟的平均负载接近或超过了 CPU 的个数，就意味着系统正在发生过载的问题，这时就得分析问题，并要想办法优化了；
- 例：假设我们在有2个 `CPU` 系统上看到平均负载为 `2.73，6.90，12.98`
  - 那么说明在过去1分钟内，系统有 `136%` 的超载 `(2.73/2=136%)`
  - 而在过去5分钟内，有 `345%` 的超载 `(6.90/2=345%)`
  - 而在过去15分钟内，有 `649%` 的超载 `(12.98/2=649%)`
  - 但从整体趋势来看，系统的负载是在逐步的降低。

5.1.8.2.2 何时需要关注平均负载

- 当平均负载高于 `CPU` 数量 `70%` 的时候，你就应该分析排查负载高的问题了。一旦负载过高，就可能导致进程响应变慢，进而影响服务的正常功能。
- 但 70% 这个数字并不是绝对的，最推荐的方法，还是把系统的平均负载监控起来，然后根据更多的历史数据，判断负载的变化趋势。
- 当发现负载有明显升高趋势时，比如说负载翻倍了，你再去做分析和调查。*

5.1.8.3 平均负载与CPU使用率

- 在实际工作中，我们经常容易把平均负载和 CPU 使用率混淆，所以在这里，我也做一个区分；
- 既然平均负载代表的是活跃进程数，那平均负载高了，不就意味着 CPU 使用率高吗?
- 我们回到平均负载的含义上来，平均负载是指单位时间内，处于可运行状态和不可中断状态的进程数；
- 所以，它不仅包括了正在使用 `CPU` 的进程，还包括等待 `CPU` 和等待 `I/O` 的进程；
- 而 `CPU` 使用率，是单位时间内 `CPU` 繁忙情况的统计，跟平均负载并不一定完全对应。比如：
  - `CPU` 密集型进程，使用大量 `CPU` 计算会导致平均负载升高，此时这两者是一致的；
  - `I/O` 密集型进程，等待 `I/O` 也会导致平均负载升高，但 `CPU` 使用率不一定很高；
  - 大量的 `CPU` 进程调度也会导致平均负载升高，此时的 `CPU` 使用率也会比较高；

5.1.8.4 平均负载案例分析实战

- 演示这三种常场景，并用 `stress、mpstat、pidstat` 等工具，找出平均负载升高的根源。
- `stress` 是 `Linux` 系统压力测试工具，这里我们用作异常进程模拟平均负载升高的场景；
- `mpstat` 是多核 CPU 性能分析工具，用来实时查看每个 `CPU` 的性能指标，及所有 `CPU` 的平均指标；
- `pidstat` 是一个常用的进程性能分析工具，用来实时查看进程的 `CPU、Mem、I/O` 等性能指标；

```
#如果出现无法使用mpstat、pidstat命令查看%wait指标建议更新下软件包
wget http://pagesperso-orange.fr/sebastien.godard/sysstat-11.7.3-1.x86_64.rpm
rpm -Uvh sysstat-11.7.3-1.x86_64.rpm
```

5.1.8.4.1 场景1-CPU密集型进程

1.第一个终端运行 `stress` 命令，模拟一个 `CPU` 使用率 `100%` 的场景：

```
[root@m01 ~]# stress --cpu 1 --timeout 600
```

2.第二个终端运行 `uptime` 查看平均负载的变化情况*

```
# 使用watch -d 参数表示高亮显示变化的区域(注意负载会持续升高)
[root@m01 ~]# watch -d uptime
17:27:44 up 2 days,  3:11,  3 users,  load average: 1.10, 0.30, 0.17
```

3.在第三个终端运行 `mpstat` 查看 `CPU` 使用率的变化情况

```
# -P ALL 表示监控所有 CPU，后面数字 5 表示间隔 5 秒后输出一组数据
[root@m01 ~]# mpstat -P ALL 5
Linux 3.10.0-957.1.3.el7.x86_64 (m01)   2019年04月29日     _x86_64_    (1 CPU)

17时32分03秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %
gnice   %idle
17时32分08秒  all   99.80    0.00    0.20    0.00    0.00    0.00    0.00    0.00
 0.00    0.00
17时32分08秒    0   99.80    0.00    0.20    0.00    0.00    0.00    0.00    0.00
 0.00    0.00

#单核CPU所以只有一个all和0
```

4.从终端二中可以看到，1分钟的平均负载会慢慢增加到 `1.00`，而从终端三中还可以看到，正好有一个 `CPU` 的使用率为 `100%`，但它的 `iowait` 为0。这说明，平均负载的升高正是由于 `CPU` 使用率为 `100%`。那么，到底是哪个进程导致了 `CPU` 使用率为 `100%` 呢？可以使用 `pidstat` 来查询

```
# 间隔 5 秒后输出一组数据
[root@m01 ~]# pidstat -u 5 1
Linux 3.10.0-957.1.3.el7.x86_64 (m01)   2019年04月29日     _x86_64_(1 CPU)

17时33分21秒   UID       PID    %usr %system  %guest    %CPU   CPU  Command
17时33分26秒     0    110019   98.80    0.00    0.00   98.80     0  stress

#从这里可以明显看到，stress 进程的 CPU 使用率为 100%。
```

5.1.8.4.2 场景2-I/O密集型进程

1.在第一个终端运行 `stress` 命令，但这次模拟 `I/O` 压力，即不停地执行 `sync`

```
[root@m01 ~]# stress  --io 1 --timeout 600s
```

2.在第二个终端运行 `uptime` 查看平均负载的变化情况：

```
[root@m01 ~]# watch -d uptime
```

```
18:43:51 up 2 days,  4:27,  3 users,  load average: 1.12, 0.65, 0.00
```

3.最后第三个终端运行 `mpstat` 查看 `CPU` 使用率的变化情况：

```
# 显示所有 CPU 的指标，并在间隔 5 秒输出一组数据
[root@m01 ~]# mpstat -P ALL 5
Linux 3.10.0-693.2.2.el7.x86_64 (bgx.com)   2019年05月07日     _x86_64_    (1 CPU)

14时20分07秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %
gnice   %idle
14时20分12秒  all    0.20    0.00   82.45   17.35    0.00    0.00    0.00    0.00
 0.00    0.00
14时20分12秒    0    0.20    0.00   82.45   17.35    0.00    0.00    0.00    0.00
 0.00    0.00

#会发现cpu的与内核打交道的sys占用非常高
```

4.导致 `iowait` 高，我们需要用 `pidstat` 来查询

```
# 间隔 5 秒后输出一组数据，-u 表示 CPU 指标
[root@m01 ~]# pidstat -u 5 1
Linux 3.10.0-957.1.3.el7.x86_64 (m01)   2019年04月29日     _x86_64_(1 CPU)
18时29分37秒   UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
18时29分42秒     0    127259   32.60    0.20    0.00   67.20   32.80     0  stress
18时29分42秒     0    127261    4.60   28.20    0.00   67.20   32.80     0  stress
18时29分42秒     0    127262    4.20   28.60    0.00   67.20   32.80     0  stress

#可以发现，还是 stress 进程导致的。
```

5.1.8.4.3 场景3-大量的进程

1.在第一个终端使用 `stress`，但这次模拟的是 `4` 个进程

```
[root@m01 ~]# stress -c 4 --timeout 600
```

2.由于系统只有 `1` 个 `CPU`，明显比 `4` 个进程要少得多，因而，系统的 `CPU` 处于严重过载状态*

```
[root@m01 ~]# watch -d uptime
19:11:07 up 2 days,  4:45,  3 users,  load average: 4.65, 2.65, 4.65
```

3.最后通过 `pidstat` 查询进程的情况：可以看出 `4` 个进程在争抢 `1` 个 `CPU` 每个进程等待 `CPU` 的时间（也就是代码块中的 `%wait` 列）高达 `75%`。这些超出 `CPU` 计算能力的进程，最终导致

`CPU` 过载。

```
# 间隔 5 秒后输出一组数据
[root@m01 ~]# pidstat -u 5 1
平均时间:   UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
平均时间:     0    130290   24.55    0.00    0.00   75.25   24.55     -  stress
平均时间:     0    130291   24.95    0.00    0.00   75.25   24.95     -  stress
平均时间:     0    130292   24.95    0.00    0.00   75.25   24.95     -  stress
平均时间:     0    130293   24.75    0.00    0.00   74.65   24.75     -  stress
```

5.1.8.5 总结

- 分析完这三个案例，我再来归纳一下平均负载与CPU
- 平均负载提供了一个快速查看系统整体性能的手段，反映了整体的负载情况。但只看平均负载本身，我们并不能直接发现，到底是哪里出现了瓶颈。所以，在理解平均负载时，也要注意：
- 平均负载高有可能是 CPU 密集型进程导致的；
- 平均负载高并不一定代表 CPU 使用率高，还有可能是 I/O 更繁忙了；
- 当发现负载高的时候，你可以使用 mpstat、pidstat 等工具，辅助分析负载的来源

5.2 计划任务

5.2.1 什么是crond

- `crond` 就是计划任务，类似于我们平时生活中的闹钟。定点执行。

5.2.2 为什么需要crond

- `crond` 主要是做一些周期性的任务。
  - 场景1: 定期备份重要的文件或数据。
  - 场景2: 促销活动，准点开启抢购接口，准点关闭抢购接口。
  - 场景3: 每分钟检测超时订单，超过30分钟未支付的订单进行取消。
  - 场景4: 每隔5分钟上各个电商平台刷取订单信息写入自己公司的系统中。

5.2.3 计划任务两大类

- 1.系统级别的定时任务： 临时文件清理、系统信息采集、日志文件切割
- 2.用户级别的定时任务： 定时备份数据，同步时间，订单超时自动取消，按时间段统计信息等等

5.2.4 计划任务基本应用

5.2.4.1 计划任务时间周期

- `Crontab` 配置文件记录了时间周期的含义

```
[root@xuliangwei ~]# vim /etc/crontab
SHELL=/bin/bash                         # 执行命令的解释器
PATH=/sbin:/bin:/usr/sbin:/usr/bin      # 环境变量
MAILTO=root                             # 邮件发给谁
# Example of job definition:
# .---------------- minute (0 - 59)     # 分钟
# |  .------------- hour (0 - 23)       # 小时
# |  |  .---------- day of month (1 - 31)   # 日期
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr # 月份
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,
sat  # 星期
# |  |  |  |  |
# *  *  *  *  *   command to be executed

# *  表示任意的(分、时、日、月、周)时间都执行
# -  表示一个时间范围段, 如5-7点
# ,  表示分隔时段, 如6,0,4表示周六、日、四
# /1 表示每隔n单位时间, 如*/10 每10分钟
```

5.2.4.2 计划任务编写范例

- 如下是已编写好的计划任务示例

```
00 02 * * * ls      # 每天的凌晨2点整执行
00 02 1 * * ls      # 每月的1日的凌晨2点整执行
00 02 14 2 * ls     # 每年的2月14日凌晨2点执行
00 02 * * 7 ls      # 每周天的凌晨2点整执行
00 02 * 6 5 ls      # 每年的6月周五凌晨2点执行
00 02 14 * 7 ls     # 每月14日或每周日的凌晨2点都执行
00 02 14 2 7 ls     # 每年的2月14日或每年2月的周天的凌晨2点执行
*/10  02 * * * ls   # 每天凌晨2点, 每隔10分钟执行一次 2:00开始 2:50结束
* * * * *  ls       # 每分钟都执行
00 00 14 2 *  ls    # 每年2月14日的凌晨执行命令
*/5 * * * *  ls     # 每隔5分钟执行一次
00 02 * 1,5,8 * ls  # 每年的1月5月8月凌晨2点执行
00 02 1-8 * *  ls   # 每月1号到8号凌晨2点执行
0 21 * * * ls       # 每天晚上21:00执行
45 4 1,10,22 * * ls # 每月1、10、22日的4:45执行
45 4 1-10 * * l     # 每月1到10日的4:45执行
```

```
3,15 8-11 */2 * * ls# 每隔两天的上午8点到11点的第3和第15分钟执行
00 23-7/1 * * * ls  # 晚上11点到早上7点之间，每小时都执行
15 21 * * 1-5 ls    # 周一到周五每天晚上21:15执行
```

5.2.5 使用crond实现计划任务

- `crond` 命令，及相关选项；

参数	含义
-e	编辑定时任务
-l	查看定时任务
-r	删除定时任务
-u	指定其他用户

5.2.5.1 场景1-定时时间同步

- 使用 `root` 用户每5分钟执行一次时间同步

```
# 1.如何同步时间
[root@xuliangwei ~]# ntpdate time.windows.com &>/dev/null

# 2.配置定时任务
[root@xuliangwei ~]# crontab -e -u root
[root@xuliangwei ~]# crontab -l -u root
*/5 * * * * ntpdate time.windows.com &>/dev/null
```

5.2.5.2 场景2-每半小时sync

- 每天的下午3,5点，每隔半小时执行一次 `sync` 命令

```
[root@xuliangwei ~]# crontab -l
*/30 15,17 * * * sync &>/dev/null
```

5.2.5.3 场景3-每天备份文件

- 每天凌晨3点做一次备份？备份 `/etc/` 目录到 `/backup` 下面
  - 1.将备份命令写入一个脚本中
  - 2.每天备份文件名要求格式：`2020-01-01_hostname_etc.tar.gz`

- 3.在执行计划任务时，不要输出任务信息
- 4.存放备份内容的目录要求只保留三天的数据

```
#1.实现如上备份需求
[root@xuliangwei ~]# mkdir /backup
[root@xuliangwei ~]# tar zcf $(date +%F)_$(hostname)_etc.tar.gz /etc
[root@xuliangwei ~]# find /backup -name “*.tar.gz” -mtime +3 -exec rm -f {}\;

#2.将命令写入至一个脚本文件中
[root@xuliangwei ~]# vim /opt/backup.sh
mkdir /backup
tar zcf $(date +%F)_$(hostname)_etc.tar.gz /etc
find /backup -name "*.tar.gz" -mtime +3 -exec rm -f {}\;

#3.配置定时任务
[root@xuliangwei ~]# crontab -l
00 03 * * * bash /root/back.sh  &>/dev/null
```

5.2.6 计划任务注意事项

- 1) 给定时任务注释
- 2) 将需要定期执行的任务写入 `Shell` 脚本中，避免直接使用命令无法执行的情况tar date
- 3) 定时任务的结尾一定要有 `&>/dev/null` 或者将结果追加重定向 `>>/tmp/date.log` 文件
- 4) 注意有些命令是无法成功执行的 `echo "123" >>/tmp/test.log &>/dev/null`
- 5) 如果一定要是用命令，命令必须使用绝对路径

5.2.7 计划任务如何备份

- 1) 通过查找 `/var/log/cron` 中执行的记录，去推算任务执行的时间
- 2) 定时的备份 `/var/spool/cron/{usernmae}`

5.2.8 拒绝特定用户使用

- `crond` 如何拒绝某个用户使用

1.使用 `root` 将需要拒绝的用户加入 `/etc/cron.deny`

```
[root@xuliangwei ~]# echo "xuliangwei" >> /etc/cron.deny
```

2.登陆该普通用户，测试是否能编写定时任务

```
[oldboy@xuliangwei ~]$ crontab -e
You (xuliangwei) are not allowed to use this program (crontab)
See crontab(1) for more information
```

5.3 运行级别

5.3.1 什么是运行级别

- 什么是运行级别，运行级别就是操作系统当前正在运行的功能级别

System V init运行级别	systemd目标名称	作用
0	runlevel0.target, poweroff.target	关机
1	runlevel1.target, rescue.target	单用户模式
2	runlevel2.target, multi-user.target	
3	runlevel3.target, multi-user.target	多用户的文本界面
4	runlevel4.target, multi-user.target	
5	runlevel5.target, graphical.target	多用户的图形界面
6	runlevel6.target, reboot.target	重启

5.3.2 调整运行级别

- `systemd` 使用 `targets` 而不是 `runlevels`
- 默认情况下，有两个主要目标:
  - `multi-user.target`： 类似于运行级别3
  - `graphical.target`： 类似于运行级别5

1.查看系统默认运行级别

```
[root@student ~]# systemctl get-default
```

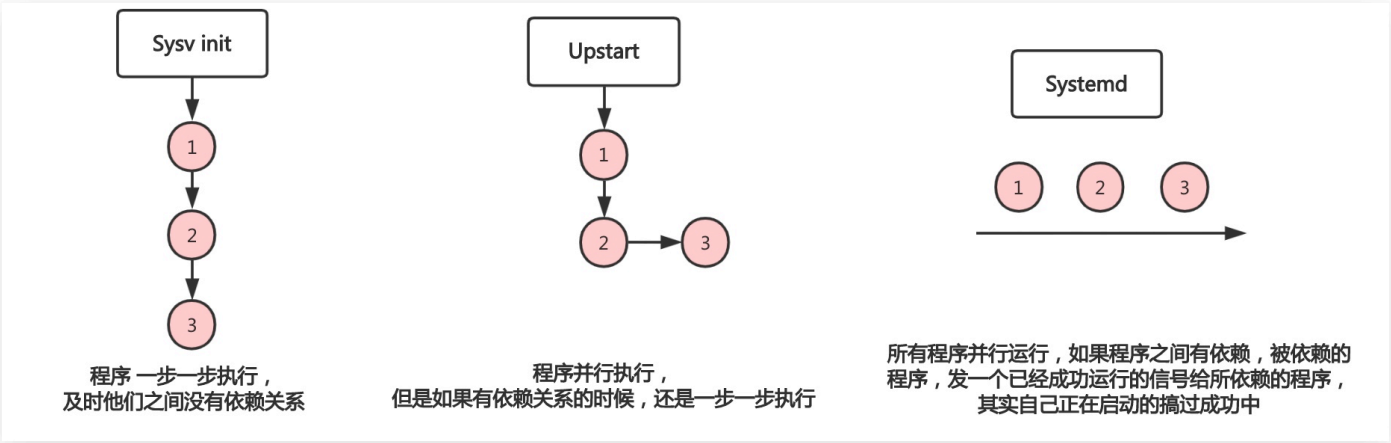
2.要设置默认目标，请运行

```
[root@student ~]# systemctl set-default TARGET.target
```

5.4 systemd

5.4.1 systemd的由来

- Linux一直以来都是采用init进程作为祖宗进程，但是init有两个缺点：
  - 1.启动时间长。Init进程是串行启动，只有前一个进程启动完，才会启动下一个进程；
  - 2.启动脚本复杂，初始化完成后系统会加载很多脚本，脚本都会处理各自的情况，这会让脚本多而复杂。
- `Centos5` 启动速度慢，串行启动过程，无论进程相互之间有无依赖关系。
- `Centos6` 启动速度有所改进，有依赖的进程之间依次启动而其他与之没有依赖关系的则并行同步启动。
- Centos7 所有进程无论有无依赖关系则都是并行启动（当然很多时候进程没有真正启动而是只有一个信号或者说是标记而已，在真正利用的时候才会真正启动。）



5.4.2 什么是systemd

- `systemd` 即为 `system daemon` 守护进程，`systemd` 主要解决上文的问题而诞生
- `systemd` 的目标是，为系统的启动和管理提供一套完整的解决方案。

5.4.3 systemd的优势

- 1、最新系统都采用 `systemd` 管理 `RedHat7、CentOS7、Ubuntu15`；
- 2、`Centos7` 支持开机并行启动服务，显著提高开机启动效率；
- 3、`Centos7` 关机只关闭正在运行的服务，而 `Centos6` 全部都关闭一次；
- 4、`Centos7` 服务的启动与停止不在使用脚本进行管理，也就是 `/etc/init.d` 下不在有脚本；
- 5、`Centos7` 使用 `systemd` 解决原有模式缺陷，比如原有 `service` 不会关闭程序产生的子进程；

5.4.4 systemd配置文件

- `/usr/lib/systemd/system/`：类似`Centos6`系统的启动脚本`/etc/init.d/`
- `/etc/systemd/system/`：类似`Centos6`系统的`/etc/rc.d/rcN.d/`
- `/etc/systemd/system/multi-user.target.wants/`
- systemd配置详解

5.4.5 systemd相关命令

systemctl管理服务的启动、重启、停止、重载、查看状态等常用命令

systemctl命令	作用
systemctl start crond.service	启动服务
systemctl stop crond.service	停止服务
systemctl restart crond.service	重启服务
systemctl reload crond.service	重新加载配置
systemctl status crond.servre	查看服务运行状态
systemctl is-active sshd.service	查看服务是否在运行中
systemctl mask crond.servre	禁止服务运行
systemctl unmask crond.servre	取消禁止服务运行

当我们使用systemctl启动一个守护进程后，可以通过sysytemctl status查看此守护进程的状态

状态	描述
loaded	服务单元的配置文件已经被处理
active(running)	服务持续运行
active(exited)	服务成功完成一次的配置
active(waiting)	服务已经运行但在等待某个事件
inactive	服务没有在运行
enabled	服务设定为开机运行
disabled	服务设定为开机不运行

static	服务开机不启动，但可以被其他服务调用启动

systemctl 设置服务开机启动、不启动、查看各级别下服务启动状态等常用命令

systemctl命令（7系统）	作用
systemctl enable crond.service	开机自动启动
systemctl disable crond.service	开机不自动启动
systemctl list-unit-files	查看各个级别下服务的启动与禁用
systemctl is-enabled crond.service	查看特定服务是否为开机自启动
systemctl daemon-reload	创建新服务文件需要重载变更

CentOS7系统，管理员可以使用 systemctl 命令来管理服务器启动与停止

```
#关机相关命令
systemctl poweroff      #立即关机，常用
#重启相关命令
systemctl reboot        #重启命令，常用
```

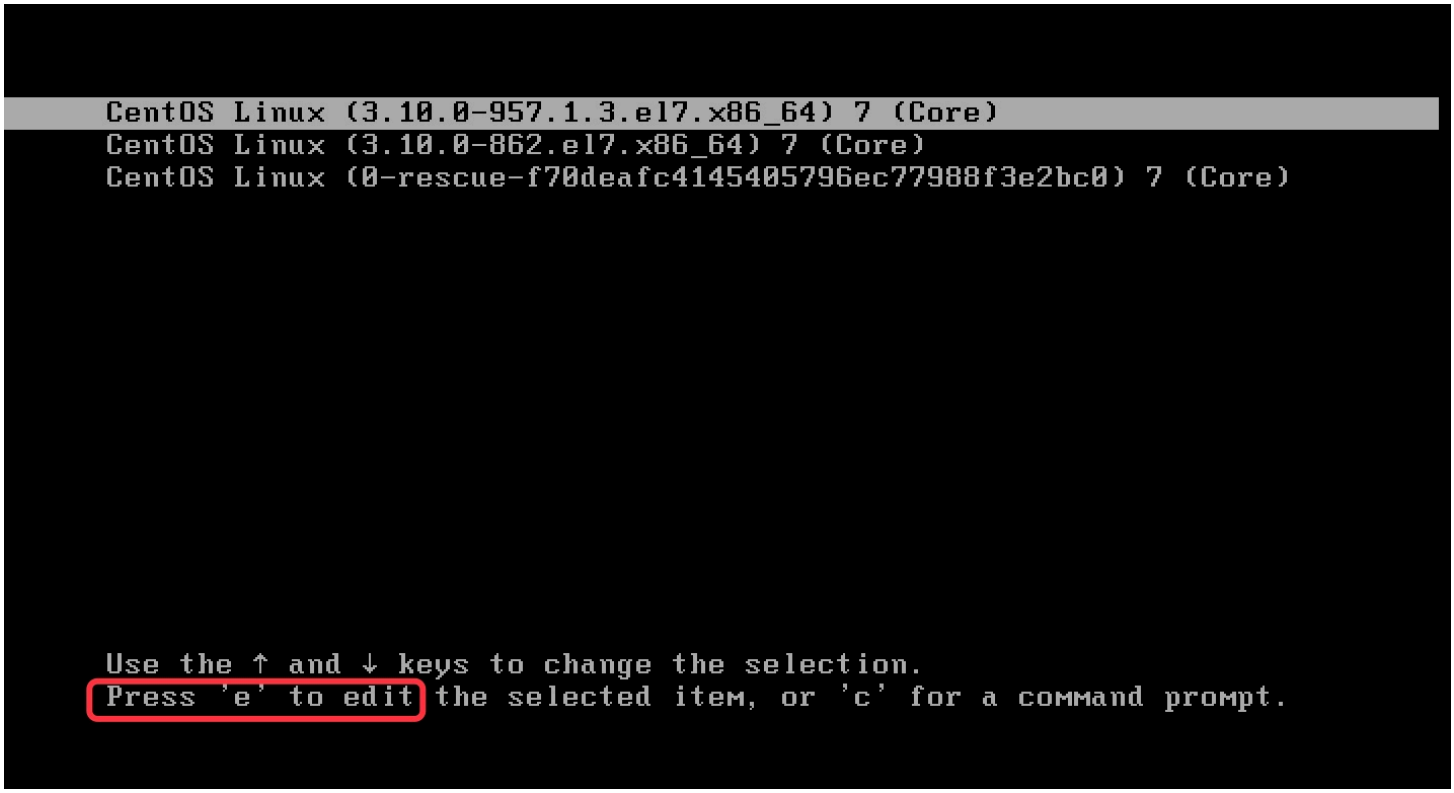
systemctl的journalctl日志

```
journalctl -n 20    #查看最后20行
journalctl -f       #动态查看日志
journalctl -p err   #查看日志的级别
journalctl -u crond #查看某个服务的单元的日志
```

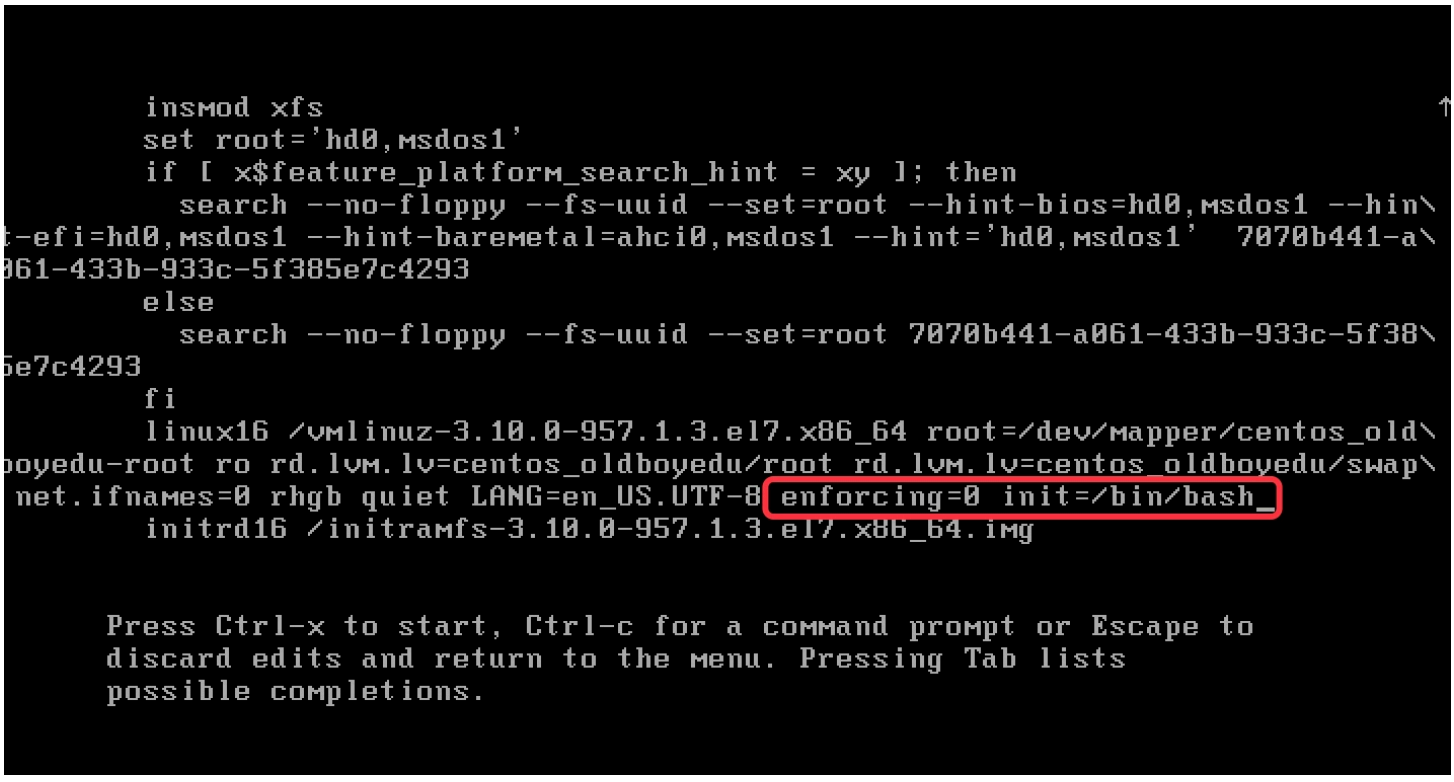
5.5 单用户模式

- 如何使用单用户模式进行变更系统密码？以Centos7系统为例：(Centos6破解方式请自行百度)

第1步：重启 `Linux` 系统主机并出现引导界面时，按下键盘上的 `e` 键进入内核编辑界面



第2步：在 `linux16` 这行的后面添加 `enforcing=0 init=/bin/bash`，然后按下 `Ctrl + X` 组合键来运行修改过的内核程序



第3步：进入到系统的单用户模式，依次输入以下命令，重启操作系统完毕，然后使用新密码来登录

```
[    1.336923] sd 0:0:0:0: [sda] Assuming drive cache: write through
bash-4.2#
bash-4.2#
bash-4.2# mount -o rw,remount / 1./默认是只读，重新挂载为读写
bash-4.2# echo "123456" | passwd --stdin root 2.使用飞交互方式修改root密码
Changing password for user root.
passwd: all authentication tokens updated successfully.
bash-4.2#
bash-4.2# exec /sbin/init _  3.执行exec  init 重新引导系统
```

5.6 Supervisor进程管理

5.6.1 Supervisor基本概述

5.6.1.1 什么是supervisor

supervisor是一个进程管理服务，主要用来将运行在前台的进程转为后台运行，并实时监控进程的状态。当出现异常时会自动将该进程拉起。

5.6.1.2 为什么要使用supervisor

其实在我们维护的Linux服务中，有不少程序没有启停脚本，并且还是前台运行，比如Python程序。当然熟悉 Linux 的朋友会考虑使用 & 符号、或 screen，将其转为后台进程。但这些方式仅仅只是将进程放置后台，当我们想要重启服务时，需要先 kill进程、然后在挂后台，比较复杂。而 Supervisor 则不同，它能实现：

1.将前台运行进程转为后台进程。

2.监控进程运行情况，如果进程异常退出，会自动重新启动进程。

3.能对服务进行启动、重启、关闭等便捷的操作。

4.提供 web ui界面，便于开发人员使用。

5.6.1.3 Supervisor基础组件

1.supervisord

主进程，负责管理进程的服务，对crash的进程重启，对进程变化发送事件通知等。

2.supervisorctl

supervisorctl命令行管理工具，可以利用它来查看被管理的进程状态，启动/停止/重启进程，获取running子进程的列表等。supervisorctl不仅可以连接到本机上的supervisord，还可以连接到远程的supervisord，当然在本机上面是通过UNIX socket连接的，远程是通过TCP socket连接的。supervisorctl和supervisord之间的通信，是通过 xml_rpc 完成的。

3.Web Server

supervisor提供了web server功能，可通过web控制进程(需要设置 [inethttpserver] 配置项)。

4.XML-RPC Interface

远程调用服务，通过HTTP协议提供的Web服务，用来控制 supervisor 以及 Supervisor 运行的进程。

5.6.2 Supervisor安装配置

5.6.2.1 Supervisor安装

```
[root@oldxu-web01 ~]# yum install supervisor -y
[root@oldxu-web01 ~]# systemctl enable supervisord
[root@oldxu-web01 ~]# systemctl start supervisord
```

5.6.2.2 Supervisor配置

2.Supervisor配置文件-[unix_http_server]

```
[unix_http_server]
file=/var/run/supervisor/supervisor.sock   ;# supervisorctl与supervisor通讯方式
;chmod=0700                 ; # socket文件的权限，默认0700
;chown=nobody:nogroup       ; # socket文件的属主与属组
;username=user              ; # 使用supervisorctl连接时，认证的用户（非必选）
;password=123               ; # 使用supervisorctl连接时，认证的密码（非必选）
```

3.Supervisor配置文件-[inet_http_server]

```
;[inet_http_server]         ; # 监听在TCP协议，WebServer需要使用，远程连接也需要使用
;port=127.0.0.1:9001        ; # 访问WebServer时使用的地址、端口
;username=user              ; # 访问WebServer的用户名称
;password=123               ; # 访问WebServer的用户名称
```

4.Supervisor配置文件-[supervisord]

```
[supervisord]           ; # 这个主要是定义supervisord这个服务端进程的一些参数的
logfile=/var/log/supervisor/supervisord.log  ; # supervisord进程日志
logfile_maxbytes=50MB   ; # 默认日志存储50m，会进行自动切割
logfile_backups=10      ; # 日志文件数量，程序启动会创建10个backup文件，用于log rotate
loglevel=info           ; # 日志级别，默认info，(debug,warn,trace)
pidfile=/var/run/supervisord.pid ; # supervisord pid文件
nodaemon=false          ; # 默认守护进程运行，true则supervisord前台运行
minfds=1024             ; # 系统最少空闲的文件描述符，低于这个值supervisor将不会启动
minprocs=200            ; # 进程最小可用文件描述符，低于这个值supervisor将不会正常启动
```

```
;umask=022                ; # 进程创建文件的掩码，默认umask 022)
;user=chrism              ; # 设置一个普通用户，后期可以通过该普通用户对supervisord进行管理
;identifier=supervisor    ; # supervisord的标识，如果有多个则需要设置不同的标识
;directory=/tmp           ; # 启动supervisord进程之前，会切换到该目录（可不配置）
;nocleanup=true           ; # false会在进程启动时，将以前子进程产生的日志文件(路径为AUTO)的清除掉。
;childlogdir=/tmp         ; # 当子进程日志路径为AUTO时，子进程日志文件存储至/tmp
;environment=KEY=value    ; # 设定环境变量，子配置文件会继承主进程定义的环境变量
;strip_ansi=false         ; # true会清除子进程日志中的所有ANSI序列。也就是日志中的\n,\t默认为false
```

4.Supervisor配置文件-[supervisorctl]

```
[supervisorctl]                       ; # 这个主要是supervisorctl的一些配置
serverurl=unix:///var/run/supervisor/supervisor.sock ; # 本地UNIX socket路径，注意和前面对应
;serverurl=http://127.0.0.1:9001      ; # supervisorctl的WebServer连接地址
;username=chris                       ; # WebServer设定的用户名称
;password=123                         ; # WebServer设定的密码
;prompt=mysupervisor                  ; # 输入用户名密码时候的提示符，默认supervisor
;history_file=~/.sc_history           ; # 历史记录，通过该文件查找历史记录
```

5.Supervisor配置文件-[program]

```
;[program:theprogramname]       ; # program是要被管理的进程，填写相应进程名称即可
;command=/bin/cat               ; # 启动进程的命令路径，可以携带参数
;process_name=%(program_name)s  ; # 进程名称，默认获取program设定的名称
;numprocs=1                     ; # 启动进程的数量
;directory=/tmp                 ; # 运行进程会切换到该目录
;umask=022                      ; # 进程掩码，默认None
;priority=999                   ; # 子进程启动和关闭的优先级，优先级低先启动，关闭时最后关闭
;autostart=true                 ; # 启动supervisord后启动被管理的子进程
;autorestart=true               ; # 当子进程挂掉后会自动重启
;startsecs=10                   ; # 子进程启动后多少秒，则认为成功启动
;startretries=3                 ; # 子进程启动失败，最大尝试启动的次数
;exitcodes=0,2                  ; # 定义退出的状态码、0或2
;stopsignal=QUIT                ; # 进程停止信号、默认TERM
;stopwaitsecs=10                ; # 向子进程发送stopsignal信号，如果超过等待时间，则强制kill子进程。
;user=chrism                    ; # 设置非root用户管理该program
;redirect_stderr=true           ; # 如果为true，则stderr的日志会被写入stdout日志文件中默认为false
;stdout_logfile=/a/path         ; # 子进程的stdout的日志路径，可以指定路径，AUTO，none等
```

```
三个选项。
;stdout_logfile_maxbytes=1MB   ; # 日志文件最大大小，默认为50M
;stdout_logfile_backups=10     ; #
;stdout_capture_maxbytes=1MB   ; #
;stdout_events_enabled=false   ; #
;stderr_logfile=/a/path        ; # 这个东西是设置stderr写的日志路径
;stderr_logfile_maxbytes=1MB   ; # 错误日志文件最大大小，默认为50M
;stderr_logfile_backups=10     ; # 错误日志文件数量
;stderr_capture_maxbytes=1MB   ; #
;environment=A=1,B=2           ; # 该子进程的环境变量，和别的子进程不进行共享
;serverurl=AUTO                ; # override serverurl computation (childutils)
```

6.Supervisor配置文件-[group],[include]

```
;[group:thegroupname]           ; # 为programs分组，方便进行统一操作
;programs=progname1,progname2   ; # 组的成员进程，用逗号分开
;priority=999                   ; # 优先级

[include]
files = supervisord.d/*.ini     ; # 包含supervisord.d目录下的所有.ini文件
```

5.6.3 Supervisor管理后台程序

5.6.3.1 Supervisor管理 Python

1.安装Python环境

```
[root@oldxu-web01 ~]# yum install openssl-devel bzip2-devel expat-devel \
gdbm-devel readline-devel sqlite-devel gcc gcc-c++ \
openssl-devel zlib zlib-devel python3 python3-devel -y
```

2.安装Django、然后手动测试服务是否正常

```
[root@oldxu-web01 ~]# pip3 install -i https://mirrors.aliyun.com/pypi/simple/ --upg
rade pip
[root@oldxu-web01 ~]# pip3 install -i https://mirrors.aliyun.com/pypi/simple/ djang
o==2.1.8
[root@oldxu-web01 ~]# django-admin.py startproject demosite
[root@oldxu-web01 ~]# cd demosite
[root@oldxu-web01 ~]# python3 manage.py runserver 0.0.0.0:8002
```

3.编写Supervisor管理 Python 的ini配置文件

```
[root@oldxu-web01 ~]# cat /etc/supervisord.d/django.ini
[program:django-python]
direct=/root/demosite
command=/bin/bash -c "python3 /root/demosite/manage.py runserver 0.0.0.0:8002"
autostart=true
autorestart=true
stdout_logfile=/var/log/django_stdout.log
stderr_logfile=/var/log/django_stderr.log
user=root
stopsignal=TERM
startsecs=5
startretries=3
stopasgroup=true
killasgroup=true
```

5.6.3.2 Supervisor管理 Java

1.安装java环境

```
[root@oldxu-web01 ~]# yum install java -y
```

2.手动启动java项目测试

```
[root@oldxu-web01 ~]# java -jar dingding-sonar-1.0-SNAPSHOT.jar --server.port=8082
```

3.编写Supervisor管理 Java 的ini配置文件，进行自动化管理

```
[root@oldxu-web01 ~]# cat /etc/supervisord.d/sonar_dingding.ini
[program:sonar-dingding]
directory=/opt
command=/bin/bash -c "java -jar dingding-sonar-1.0-SNAPSHOT.jar --server.port=8082"
autostart=true
autorestart=true
stderr_logfile=/var/log/sonar_dingding_stderr.log
stdout_logfile=/var/log/sonar_dingding_stdout.log
user = root
stopsignal=QUIT
startsecs=10
startretries=5
stopasgroup=true
killasgroup=true
```

5.6.4 SupervisorWeb页面

1.编辑supervisor主配置文件

```
[inet_http_server]
port=0.0.0.0:9001
username=oldxu
password=123
```

2.通过浏览器访问9001端口