

# Linux进程管理

- 1 作者：牟建波 (1353429820@qq.com)
- 2 时间：2025-03-20
- 3 描述：日常学习笔记

## 1.进程和内存管理

### 1.1 进程和线程概念

- 进程是正在运行的程序，是操作系统管理和调度的基本单位。在Linux系统中，每个进程都有唯一的进程ID( PID )用于标识和管理
- 线程是进程内的最小执行单位，同一进程中多个线程共享资源，但可以并发执行，提高程序效率

☁ 进程的特点：

- 动态性：进程是程序的一次执行过程，会随时变化
- 并发性：任何进程都可以同其他进程一起并发执行
- 独立性：进程是系统进行资源分配和调度的一个独立单位
- 结构性：进程由程序、数据和进程控制块三部分组成

线程的特点：

- 轻量级：线程的上下文切换比进程更快，占用资源更少
- 共享资源：线程共享进程的代码段、数据段、打开的文件等，但有自己的栈和寄存器
- 独立调度：线程可以被CPU独立调度，实现并发执行
- 并发性高：适用于I/O密集型任务，如Web服务器、数据库查询

☁ 进程分类：守护进程、前台进程

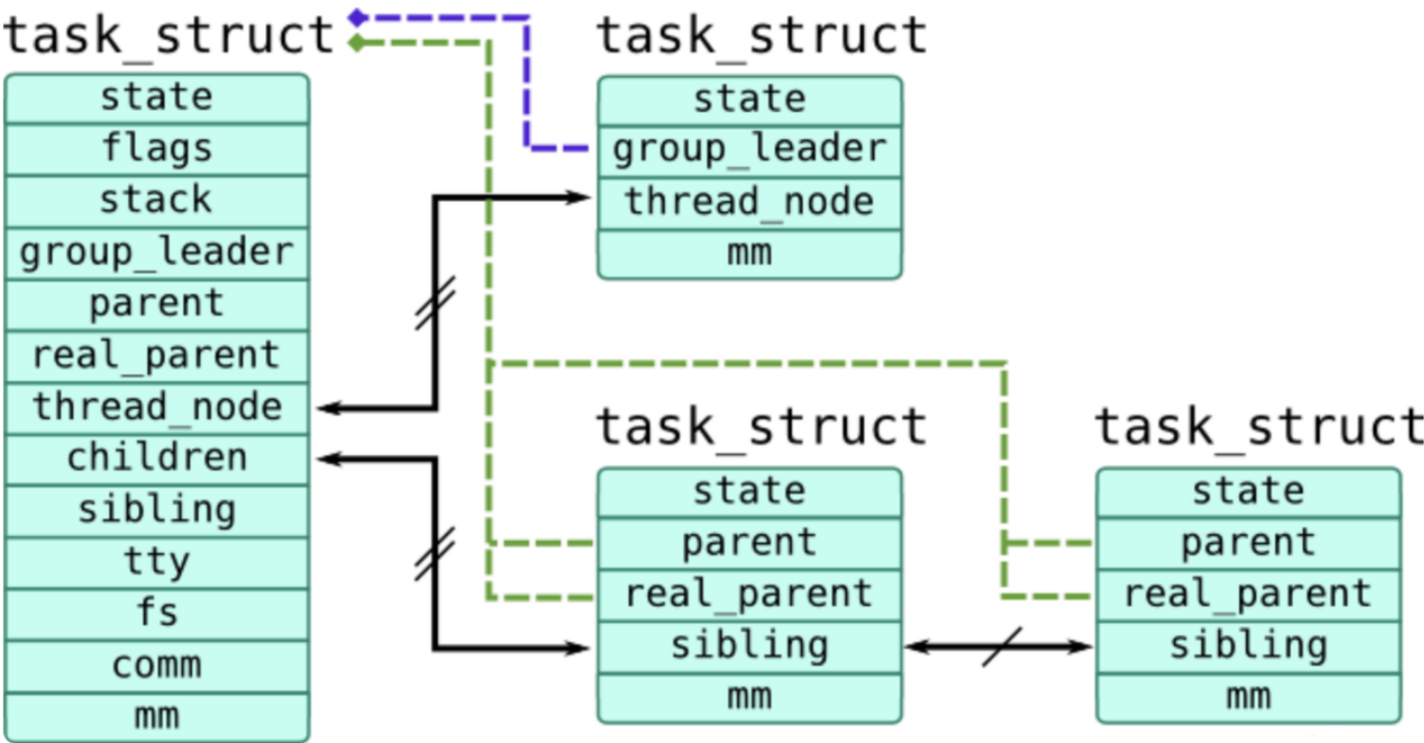
- 守护进程(daemon)：系统引导过程中启动的进程
- 前台进程：通过终端启动的进程

对比项	进程 (Process)	线程 (Thread)
定义	独立运行的程序实例	进程内的执行单元
资源	进程拥有独立的内存、文件、变量等资源	线程共享进程资源，如全局变量、文件句柄等
调度	由操作系统管理	由操作系统或进程管理
通信	需要 IPC 机制（管道、共享内存等）	线程间可直接访问进程数据
切换开销	进程切换开销大	线程切换开销小
崩溃影响	进程崩溃不影响其他进程	线程崩溃可能导致整个进程崩溃
适用场景	适用于独立任务	适用于需要高并发的任务（如 Web 服务器）

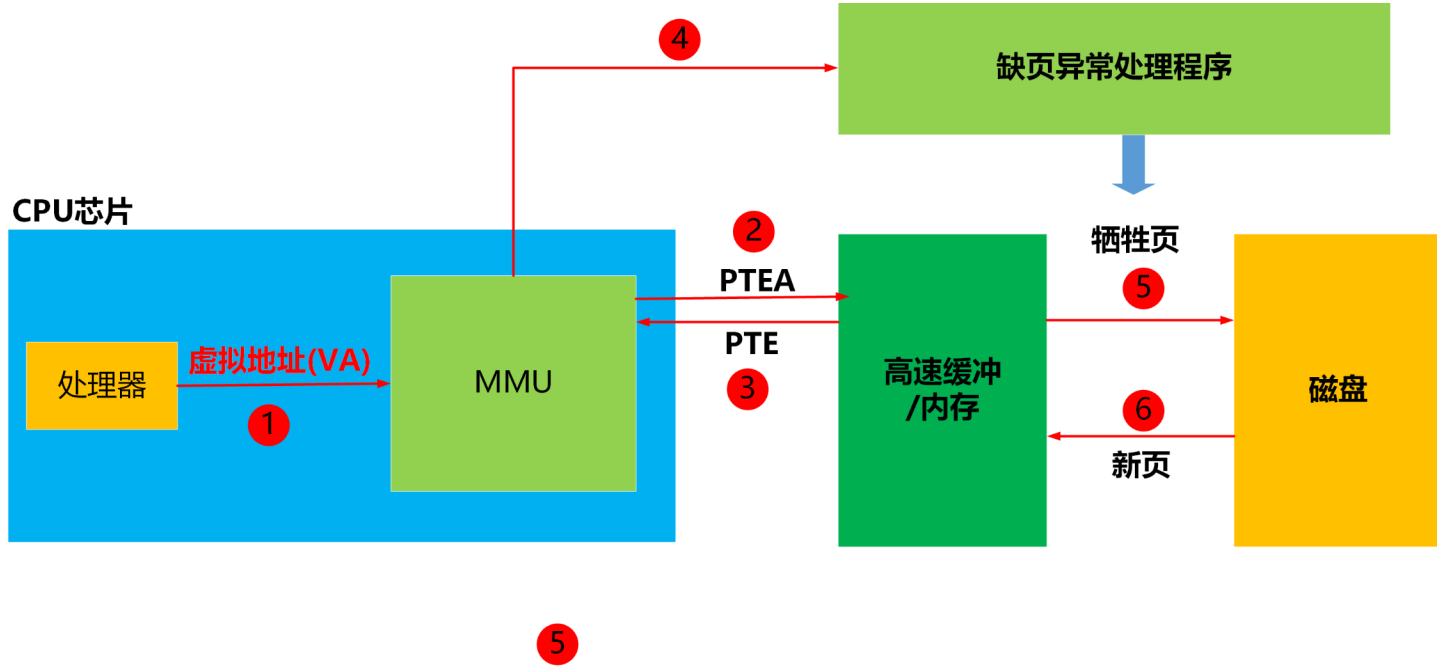
```
1  grep -i threads /proc/PID/status      # 查看进程中的线程
```

## 1.2 进程结构

☁ 内核把进程存放在任务队列(task list)的双向列表中，链表中的每一项都是类型为 `task_struct` 的结构体，称为进程控制块 `PCB` (Processing Control Block)，PCB中包含一个具体进程的所有信息

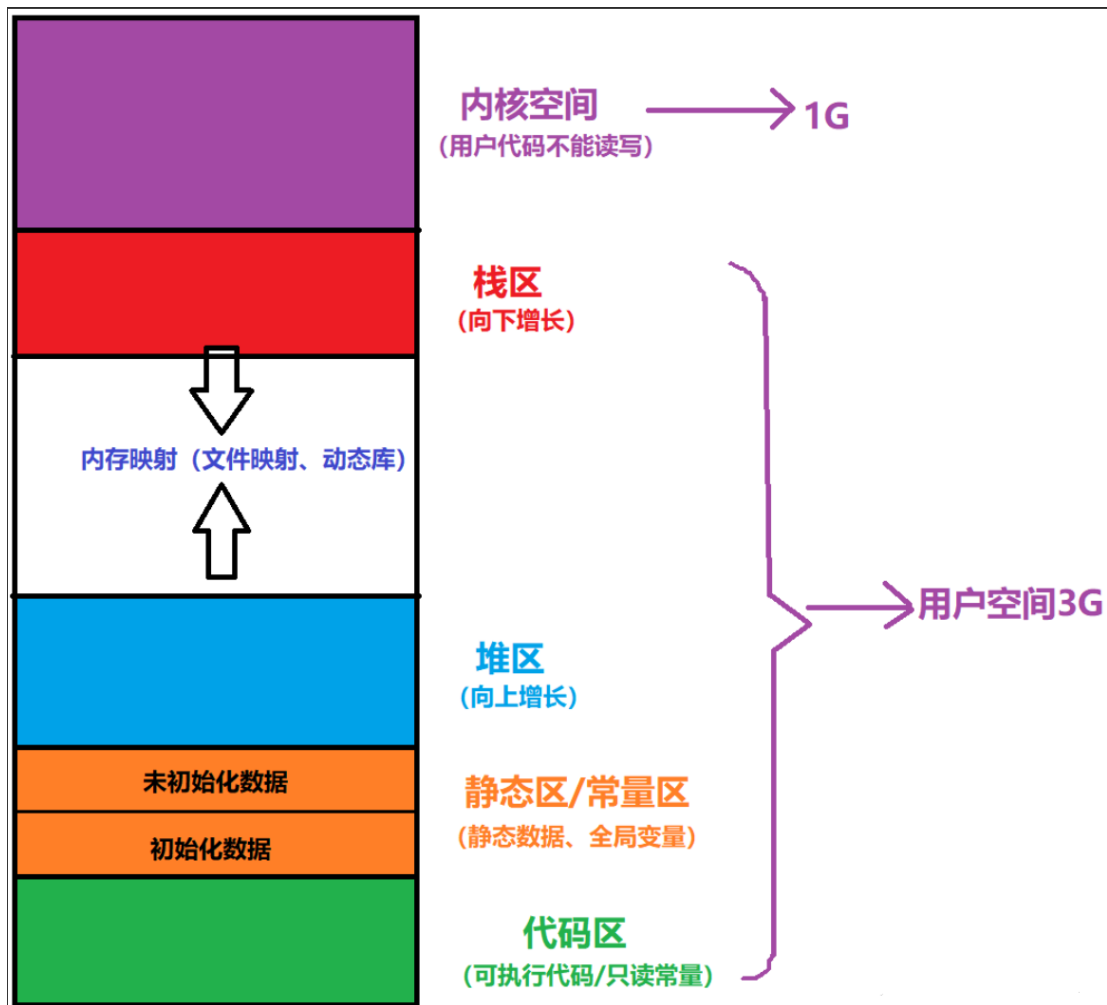


### 1.3 物理地址空间与虚拟地址空间



- ☁ 物理地址空间：物理地址是内存单元在实际RAM中的位置，由CPU和内存控制器直接访问
- 虚拟地址空间：虚拟地址是进程看到的逻辑地址，它需要映射到物理地址才能访问内存
- MMU(内存控制单元)：Memory Management Unit，负责虚拟地址转换为物理地址
  - 程序在访问一个内存地址指向的内存时，CPU不会直接把这个地址送到内存总线上，而是被送到MMU，然后把这个内存地址映射到实际的物理内存上，然后通过总线再去访问内存，程序操作的地址称为虚拟内存地址
  - TLB(翻译后备缓冲区)：Translation Lookaside Buffer，用于保存虚拟地址和物理地址映射关系的缓存

### 1.4 用户和内核空间



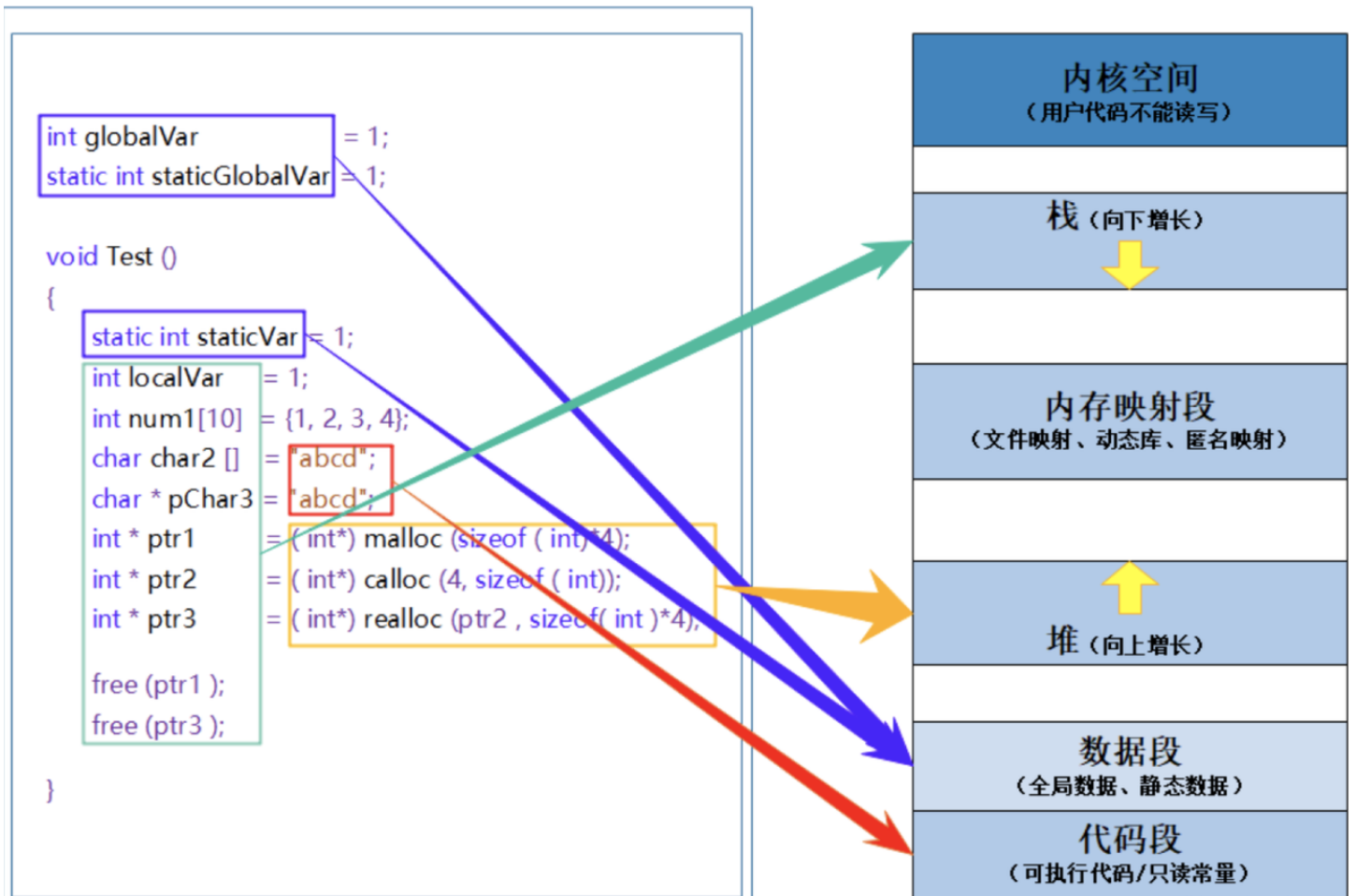
☁ 内核空间 ( Kernel Space )：运行操作系统核心代码，可访问所有资源

用户空间 ( User Space )：运行用户进程，不能直接访问硬件，通过系统调用与内核交互

用户态和核心态切换：主要通过系统调用、硬件中断、异常触发

Linux进程地址空间：

- 32位系统： 1GB内核空间 + 3GB用户空间
- 64位系统： 128TB内核空间 + 128TB用户空间



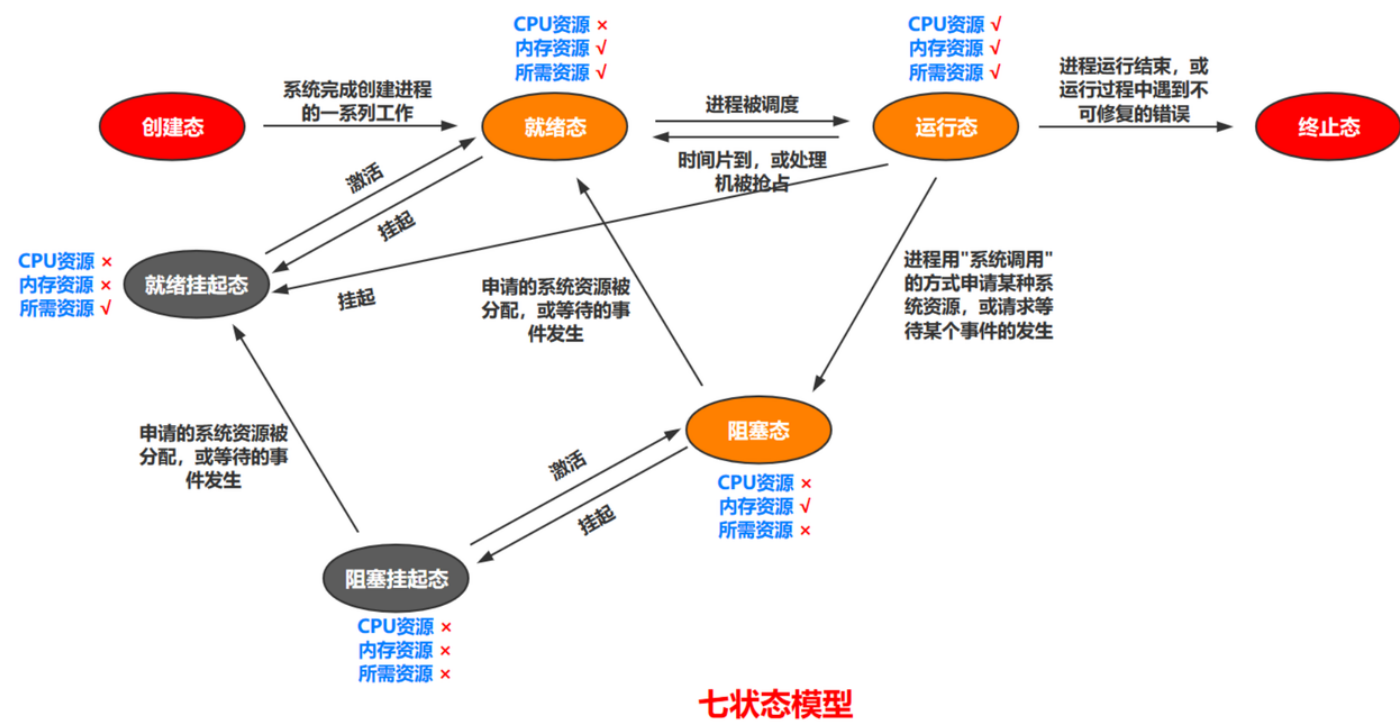
在 Linux 进程的虚拟地址空间中，主要包括以下部分（从低地址到高地址）：

内存段	地址增长方向	作用
代码段 (Text Segment)	固定	存放可执行代码，通常是只读的
数据段 (Data Segment)	固定	存放全局变量和静态变量
堆 (Heap)	向上增长	运行时动态分配的内存，如 <code>malloc()</code>
内存映射段 (Memory-Mapped Segment)	固定	共享库 ( <code>.so</code> )、文件映射、匿名映射
栈 (Stack)	向下增长	存放局部变量、函数调用信息
内核空间 (Kernel Space)	固定	内核代码、内核数据结构、设备驱动程序等

#### ☁ 内存常见问题：

- 内存泄漏(Memory Leak)：指程序中用`malloc`或`new`申请了一块内存，但是没有用`free`或`delete`将内存释放，导致这块内存一直处于占用状态
- 内存溢出(Memory Overflow)：指程序申请了10M的空间，但在这个空间写入10M以上字节的数据就是溢出
- 内存不足(Out Of Memory)：内存用完了，这种情况在java程序中比较常见。系统会选一个评分低的进程将之杀死从而腾出内存

# 1.5 进程状态



进程状态	描述	ps 命令状态符号
创建态	进程正在创建，尚未准备好运行	N/A (不可见)
就绪态	进程已具备运行条件，但等待 CPU	R (Running or Runnable)
运行态	进程正在 CPU 上执行	R (Running or Runnable)
阻塞态	进程等待 I/O 或资源，不可执行	D (Uninterruptible sleep)
终止态	进程执行结束，等待资源释放	Z (Zombie)
就绪挂起态	进程已准备好运行，但被挂起	T (Stopped by job control)
阻塞挂起态	进程等待资源，并被挂起	S (Interruptible sleep)

状态符号	含义
R	运行或就绪 (Running or Runnable)
S	可中断睡眠 (Sleeping, 等待事件发生)
D	不可中断睡眠 (Uninterruptible Sleep, 通常为 I/O 等待)
T	停止 (Stopped, 可能是被 SIGSTOP 信号暂停)
Z	僵尸进程 (Zombie, 进程已终止但未被回收)
X	已退出 (Dead, 极少见)

☁ 僵尸进程：父进程结束前，子进程不关闭，杀死父进程可以关闭僵尸态的子进程

```
1 echo $BASHPID      # 查看bash的PID值(假如为1974)，状态为Ss+(可中断睡眠+会话首领+前台进程组)
2 echo $PPID         # 查看bash的父进程PID值(假如为1973)
3 kill -19 1973      # 将父进程1973设置为停止状态T
4 kill -9 1974       # 杀死子进程，使其进入僵尸态Z
5 ps aux             # 查看子进程1974变成僵尸态Zs(僵尸状态+会话首领)
6 kill -18 1973      # 解决僵尸进程：恢复父进程1973
7 kill -9 1973       # 解决僵尸进程：杀死父进程1973
8 ps aux             # 僵尸进程不存在了
```

jimbo	1973	0.2	0.3	126548	5448	?	S	06:25	0:00	sshd: jimbo@pts/2
jimbo	1974	0.1	0.2	226284	5108	pts/2	Ss+	06:25	0:00	-bash

jimbo	1973	0.0	0.3	126548	5448	?	T	06:25	0:00	sshd: jimbo@pts/2
jimbo	1974	0.0	0.0	0	0	?	Zs	06:25	0:00	[bash] <defunct>

☁ 孤儿进程：子进程退出前，父进程先退出，导致子进程成为孤儿进程。孤儿进程会被PID=1的systemd进程( `init` )收养，成为systemd的子进程

```
1 # 运行后，父进程立即退出，子进程还在sleep(10)，此时子进程的父进程变成init，并继续执行
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 int main()
7 {
8     pid_t pid = fork();
9     if (pid > 0)
10     {
11         printf("父进程退出，子进程变成孤儿进程\n");
12         exit(0);    // 父进程退出
13     }
14     else if (pid == 0)
15     {
16         sleep(10);  // 子进程继续运行
17         printf("我是孤儿进程，PID=%d，PPID=%d\n", getpid(), getppid());
18     }
19     return 0;
20 }
```

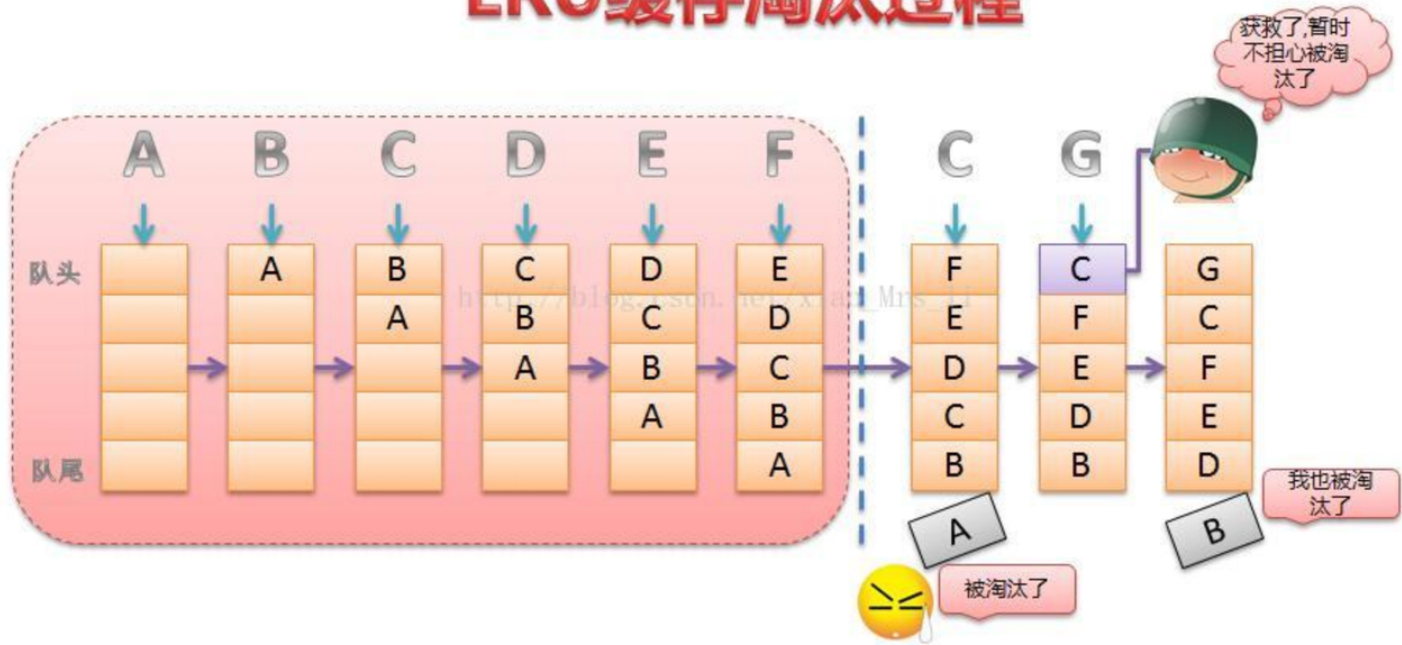


## 1.6 LRU算法

☁ LRU(近期最少使用算法): Least Recently Used算法, 用于释放内存, 喜新厌旧

- 算法思想: 队列中没有就加入队列, 有就将其放置最上面, 超出队列长度就淘汰

### LRU缓存淘汰过程



## 1.7 IPC进程间通信

在 Linux 中, **进程间通信 (IPC)** 允许多个进程共享数据、同步操作或协作完成任务。

**IPC 的主要方式:**

IPC 方式	特点	适用场景
管道 (Pipe)	单向通信, 基于文件描述符	父子进程通信
有名管道 (FIFO)	支持无亲缘关系进程通信	多个进程间数据交换
消息队列 (Message Queue)	基于消息, 可以存储消息	复杂的进程通信
共享内存 (Shared Memory)	速度快, 多个进程直接访问同一块内存	高速数据交换
信号量 (Semaphore)	主要用于进程同步, 避免资源竞争	进程同步
信号 (Signal)	内核向进程发送异步通知	进程终止、进程控制
Socket 套接字	适用于网络或本地进程通信	分布式系统、网络编程



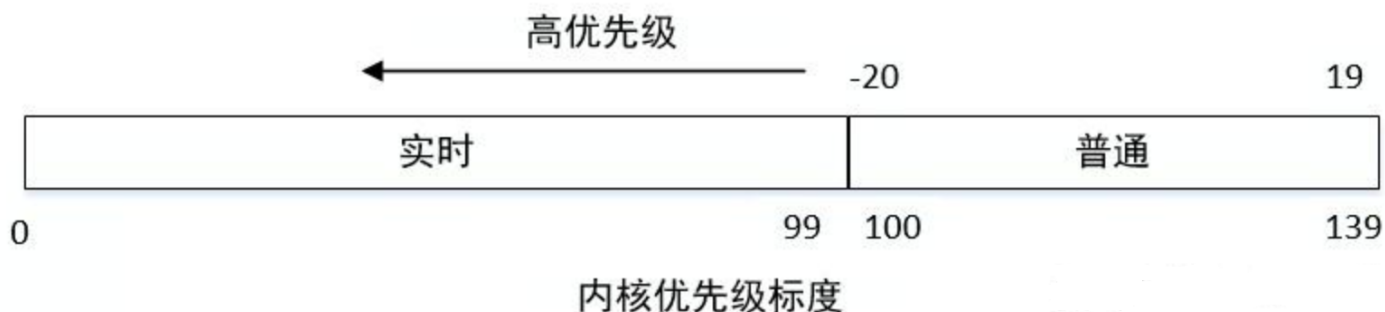
☁ 同一主机通信：管道、Socket套接字、共享内存、信号、信号量

- Socker=IP+端口号

不同主机通信：RPC远程过程调用(Remote Procedure Call)、MQ消息队列(Message Queue)

```
1 # 管道文件实现IPC
2 mkfifo /data/test.fifo # 在A终端创建管道文件，写入数据
3 ll /data/test.fifo
4 cat > /data/test.fifo
5
6 cat /data/test.fifo # 在B终端可以从文件中读取数据
```

## 1.8 进程优先级



☁ 进程真实优先级 =  $PRI + NI$  (值越小, 优先级越高)

- **PRI** : 进程优先级, 默认值是 80
- **NI** : 进程优先级修正数据, 默认值是 0 , nice: [-20, 19]
  - 普通用户: 只能提高nice值 (降低优先级)
  - root用户: 可以降低nice值 (提高优先级)
- Linux进程优先级极值: [60, 99]

## 2. 进程管理与性能工具

### 2.1 进程管理工具

```
1  # 1.pstree: 显示进程树
2  选项:
3      -p: 显示PID
4      -T: 不显示线程, 默认显示线程
5      -u: 显示用户切换
6
7  pstree -p                # 常用显示进程PID树
8
9
10 # 2.ps: 显示进程信息
11 选项:
12      a: 包括所有终端中的进程
13      u: 显示进程所有者信息
14      x: 包括不链接终端的进程
15
16 ps aux                   # 显示进程信息
17 ps aux | grep xxx       # 查找xxx进程信息
18
19
20 # 3.prtstat: 显示单个进程详细信息 (不好用)
21 选项:
22      -r: raw格式排列整齐显示
23 prtstat -r PID          # 显示PID进程详细信息
24
25
26 # 4.nice/renice: 调整进程优先级
27 启动进程时设置优先级: nice -n [优先级] 命令
28 调整已运行进程优先级: renice [新优先级] -p [PID]
29
30 nice -n 10 ./my_program  # 以nice值10运行my_program
31 nice -n -5 ./my_program  # 以较高优先级-5运行, 需root权限
32 renice 5 -p 1234         # 将PID为1234的进程nice值调整为5
33 renice -10 -p 5678       # 将PID为5678的优先级提高, 需要root权限
34
35
36 # 5.ps/pgrep/pidof: 搜索进程
37 ps aux | grep 进程名     # 适合查对应名字进程 (推荐使用)
38 pgrep 进程名             # 适用于常规进程, 简单快捷
39 pidof 进程名             # 查找二进制程序的PID
40
41
42 # 6.lsof: 查看进程打开文件
43 lsof                     # 列出所有打开的文件
44 lsof -p 1234             # 按进程PID查找
45 lsof -u root             # 按用户查找
46 lsof -i :80              # 按端口查找
47
48 # 使用lsof恢复正在使用中的误删除文件
```

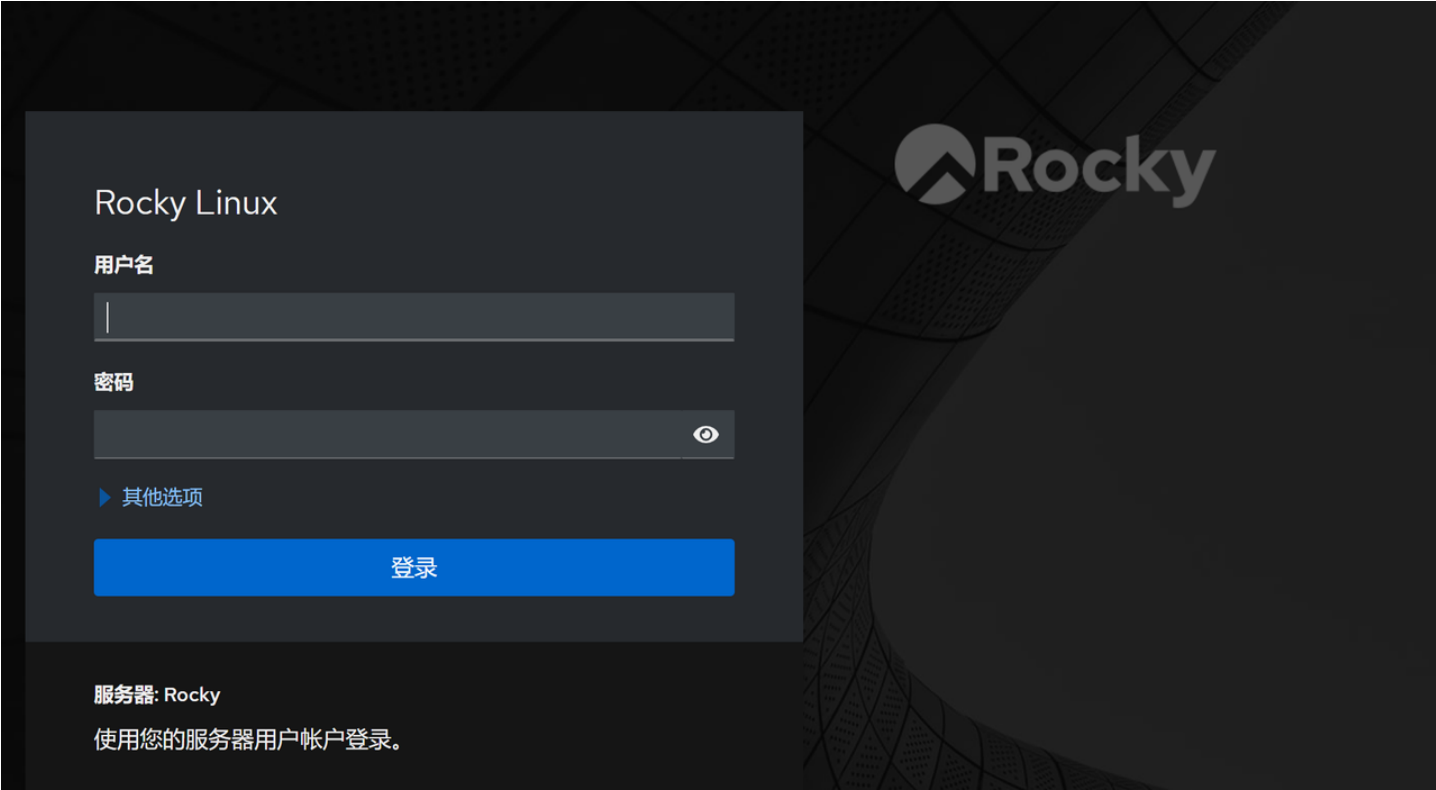
```
48  lsof | grep deleted # 确定被删除但仍占用的文件
49                        # 例如: nginx 1234 4w /var/log/nginx/access.log(deleted)
50
51  ls -l /proc/1234/fd | grep deleted      # 找回误删除的文件, 假如这里文件描述符是4
52
53  cp /proc/1234/fd/4 /root/recovered_access.log      # 使用cp/cat进行恢复
54  cat /proc/1234/fd/4 > /root/recovered_access.log  # 恢复后可以手动重命名到原位置
55
56
57  # 7.kill: 进程信号发送
58  常用信号: 使用时直接用数字代替    kill -n <PID>
59      1) SIGHUP: 挂起(重载配置文件)
60      2) SIGINT: 终止进程(ctrl+c)
61      9) SIGKILL: 强制终止进程(无法拦截)
62      15) SIGTERM: 默认终止信号(进程可捕获)
63      18) SIGCONT: 恢复进程
64      19) SIGSTOP: 暂停进程(ctrl+z)
65
66  kill -9 1234      # 删除PID为1234的进程
67  kill -19 1234     # 暂停PID为1234的进程
68  kill -18 1234     # 恢复PID为1234的进程
```

## 2.2 进程性能工具

```
1  # 1.uptime: 负载查询
2  说明: 系统平均负载(1、5、15分钟的平均负载, 一般不超过1, 超过5时建议报警, 系统性能较差)
3  uptime      # 08:42:36 up 48 min,  1 user,  load average: 0.56, 0.28, 0.20
4
5
6  # 2.mpstat: 显示CPU相关统计
7  sudo yum install -y sysstat # 安装mpstat工具
8  mpstat      # 显示CPU相关统计信息
9
10
11 # 3.top/htop: 查看进程实时状态 (htop更高级)
12 top      # 按q退出
13 top -d 时间      # 指定刷新时间, 默认3s
14 sudo yum install -y htop      # 安装htop工具
15 htop      # 按q退出, 功能按键直接显示
16
17
18 # 4.free: 显示内存空间
19 free -h      # 以易读格式显示
```

```
20
21
22 # 5.pmap: 显示进程对应的内存映射
23 sudo pmap 1 # 显示PID为1进程的内存占用情况
24
25
26 # 6.vmstat: 显示虚拟内存信息
27 vmstat # 显示虚拟内存信息
28
29
30 # 7.iostat: 统计CPU和设备IO信息
31 iostat # 显示CPU和设备IO信息
32
33
34 # 8.iotop: 监视硬盘
35 sudo yum install -y iotop # 安装iotop工具
36 sudo iotop # 监视硬盘信息, 按q退出
37 sudo iotop -d 10 # 10s刷新一次
38
39
40 # 9.iftop: 监视网络带宽使用信息
41 sudo yum install -y iftop # 安装iftop工具
42 sudo iftop # 显示网络带宽使用信息
43
44
45 # 10.nload: 显示网络实时吞吐量
46 nload # 显示网络实时吞吐量, 按q退出
47
48
49 # 11.nethogs: 显示进程网络带宽使用信息
50 sudo yum install -y nethogs # 安装nethogs工具
51 sudo nethogs # 显示进程网络带宽使用信息
52
53
54 # 12.iptraf-ng: 网络监视工具
55 sudo yum install -y iptraf-ng # 安装iptraf-ng工具
56 sudo iptraf-ng # 显示iptraf-ng操作界面
57
58
59 # 13.dstat: 系统资源统计工具
60 sudo yum install -y dstat # 安装dstat工具
61 dstat # 使用dstat工具
62
63
64 # 14.glances: 综合监控工具
65 sudo yum install -y glances # 安装glances工具
66 glances # 使用glances工具
```

```
67
68
69 # ★15.cockpit: 图形化资源网站监控工具
70 sudo yum install -y cockpit # 安装cockpit工具
71 sudo systemctl enable --now cockpit.socket
72 https://<cockpit主题>:9090 # https://192.168.32.133:9090
```



jimbo@Rocky

管理员模式

帮助

会话

搜索

jimbo@Rocky:~ 这里面还自带了终端

字体大小

-

16

+

外观

黑色

重置

系统

概览

日志

网络

用户账户

服务

工具

内核转储

软件更新

应用程序

诊断报告

终端

SELinux

[jimbo@Rocky ~]\$

jimbo@Rocky

管理员模式

帮助

会话

搜索

Mbps 传输

Mbps 接收

防火墙 禁用

编辑规则和区域

0 个已启用的安全区域

接口

添加 VPN

添加绑定

添加绑定

添加网桥

添加 VLAN

名称

IP 地址

发送

接收

ens160

192.168.32.133/24

96.6 Kbps

6.45 Mbps

网络日志

查看所有日志

2025年3月24日

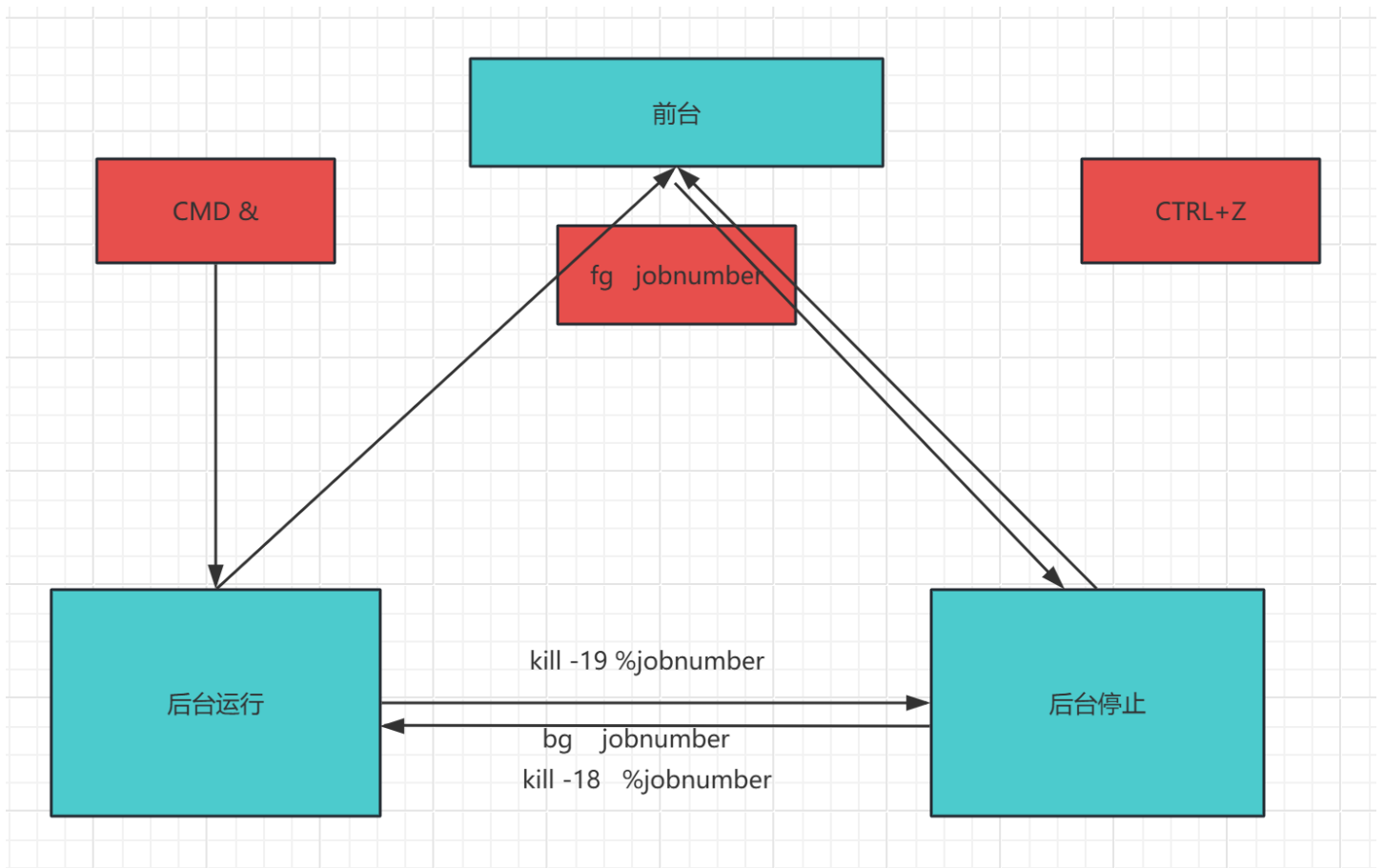
13:54 <info> [1742795668.0111] dhcp4 (ens160): state changed new lease, address=192.168.32.133

NetworkManager

13:39 <info> [1742794768.0091] dhcp4 (ens160): state changed new lease, address=192.168.32.133

NetworkManager

## 2.3 进程作业管理



☁ 前台作业：通过终端启动，启动后一直占据终端

后台作业：通过终端启动，启动后转入后台运行，释放终端

```

1  # 1.jobs: 显示当前作业
2  jobs -l                                # 显示当前作业ID
3  [1]+  Running ping 8.8.8.8 &
4  [2]-  Stopped nano file.txt
5  # 说明: [1]+是作业编号, Running是作业状态, &表示作业在后台运行
6
7
8  # 2.后台运行作业
9  ping 8.8.8.8 > ping.long &           # 直接启动后台作业
10 nano file.txt 按ctrl+z                # 使用ctrl+z暂停作业, 从而前台作业移到后台暂停
11 bg %2                                # 将后台暂停的作业移动到后台运行, 2是jobs显示的作业ID
12
13
14 # 3.将后台作业恢复到前台
15 fg %1                                # 让作业在前台运行, 1是jobs显示的作业ID
16
17
18 # 4.终止作业
19 kill %1                                # 终止后台进程
20 kill -9 %1                            # 强制终止后台进程
  
```



```
21
22
23 # 5.后台作业保持运行: nohup、disown让作业不受终端影响
24 nohup ping 8.8.8.8 > ping.log &      # ping不会因为终端关闭而结束
25 disown -h %1                          # 如果作业已经运行, 使用disown让其脱离终端
```

## 2.4 进程并行运行

```
1 方法一: command1 & command2 & command3 &
2 ping -c 5 8.8.8.8 & ping -c 5 1.1.1.1 &      # 同时对8.8.8.8和1.1.1.1进行
ping测试
3
4
5 方法二: command1; command2; command3
6 echo "First"; sleep 2; echo "Second"
7
8
9 方法三: command1 && command2、command1 || command2
10 mkdir test_dir && test_dir                    # &&: 前一个命令成功时执行下一个命
令
11 ping -c 1 8.8.8.8 || echo "Network is down"  # ||: 前一个命令失败时执行下一个命
令
```

## 3.任务计划

### 3.1 一次性任务

☁ `at` : 指定时间点, 执行一次性任务

`batch` : 系统自行选择空闲时间去执行指定的任务 (基本不用)


```
1 # 1.at命令
2 sudo yum install -y at      # 安装at命令
3 sudo systemctl start atd    # 启动atd(任务调度守护进程)
4 sudo systemctl enable atd
5
6 # at命令使用
```

```

7  at <时间>                                # 输入要执行的命令，按ctrl+d结束输入
8  at now                                    # 立刻执行
9  at now + 10 minutes                      # 10分钟后执行
10 at now + 1 day                          # 1天后执行
11 at 14:30                                # 指定时间执行
12 at 9:00 2025-03-30                      # 指定日期执行
13
14 # 查看待执行任务
15 atq
16
17 # 删除任务
18 atrm <任务编号>
19
20 # 2.batch命令
21 batch                                    # 输入要执行的命令，按ctrl+d结束输入

```

## 3.2 周期性任务

 **cron**：定时任务调度工具，用于定期执行任务

```

1  sudo systemctl start cron                # 启动cron
2  sudo systemctl enable cron              # 开机自启动cron
3  sudo systemctl restart cron            # 重启cron
4  sudo systemctl status cron             # 查看cron状态
5
6  # cron格式
7
8  ┌────────── 分钟 (0 - 59)
9  │ ┌──────── 小时 (0 - 23)
10 │ │ ┌────── 日期 (1 - 31)
11 │ │ │ ┌── 月份 (1 - 12)
12 │ │ │ │ ┌ 星期 (0 - 7) (0 或 7 代表星期天)
13 │ │ │ │ │
14 │ │ │ │ │ 需要执行的命令
15
16 * * * * * echo "Hello, World!" >> /tmp/cron_test.log # 每分钟
    向/tmp/cron_test.log追加内容
17 0 * * * * /path/to/script.sh                    # 每小时整点执行
    script.sh脚本
18 0 3 * * * /usr/bin/backup.sh                    # 每天凌晨3点执行
    backup.sh备份脚本

```

```
19 0 6 * * 1 /path/to/task.sh # 每周一早上六点执行
    task.sh脚本
20 0 0 1 * * /usr/local/bin/cleanup.sh # 每月一号0:00执行
    cleanup.sh脚本
21 0 0 1 1 * echo "Happy New Year!" >> /tmp/new_year.log # 每年1月1日在
    new_year.log追加内容
```