

进程

补vim上面的一些操作:

- 命令模式下`ctrl+v`,然后通过`hjkl`来选中区域,然后输入`A`输入`//`最后按`esc`来进行批量化注释
- 命令模式下`ctrl+v`,然后通过`hjkl`来选中区域,然后按`d`就可以批量化删除了

补查看进程的方式:

- `ls -d /proc`(查看pid为1286的进程`ls -al /proc/1286`)

```
[xifeng@VM-16-14-centos lesson8]$ ls /proc
1      1412    2       253     278     43      6      9448    execdomains  kpageflags
10     1443    20      2532    279     4362    662    976     loadavg
1040   1484    21      2533    2976    44      663    acpi     filesystems  locks
1043   15     2154    2536    30      45      664    buddyinfo fs           mdstat
1046   15835  22      254     31      46      665    bus      interrupts  meminfo
108    16     23     2544    32      508     666    cgroups  iomem       misc
11     1604    24      257     33      543     669    cmdline  ioports     modules
1282   17     245     2634    3469    566     6971   consoles irq          mounts
1286   18     24505  26355  3693    568     7      cpuinfo  kallsyms    mtrr
13     19     251     26362  372     569     736    crypto   kcore       net
1300   192    2516    26363  395     571     8      devices  keys        pagetypeinfo
1304   19480  252     26452  4       576     9      diskstats key-users   partitions
1305   19487  2520    26464  401     59      9398   dma      kmsg        sched_debug
14     19497  2528    268     41      592     9447   driver   kpagecount  schedstat
```

- 当进程开始的时候就会在`/proc`目录里面创建一个目录,进程结束后会自动消失

```
[xifeng@VM-16-14-centos lesson8]$ ls /proc/4136 -al
total 0
dr-xr-xr-x  9 xifeng xifeng 0 Mar  9 13:36 .
dr-xr-xr-x 112 root   root   0 Jan 22 13:34 ..
dr-xr-xr-x  2 xifeng xifeng 0 Mar  9 13:38 attr
-rw-r--r--  1 xifeng xifeng 0 Mar  9 13:38 autogroup
-r-----  1 xifeng xifeng 0 Mar  9 13:38 auxv
-r--r--r--  1 xifeng xifeng 0 Mar  9 13:38 cgroup
--w-----  1 xifeng xifeng 0 Mar  9 13:38 clear_refs
-r--r--r--  1 xifeng xifeng 0 Mar  9 13:38 cmdline
-rw-r--r--  1 xifeng xifeng 0 Mar  9 13:38 comm
-rw-r--r--  1 xifeng xifeng 0 Mar  9 13:38 coredump_filter
-r--r--r--  1 xifeng xifeng 0 Mar  9 13:38 cpuset
lrwxrwxrwx  1 xifeng xifeng 0 Mar  9 13:36 cwd -> /home/xifeng/lesson8
lrwxrwxrwx  1 xifeng xifeng 0 Mar  9 13:36 exe -> /home/xifeng/lesson8/test
dr-x-----  2 xifeng xifeng 0 Mar  9 13:36 fd
dr-x-----  2 xifeng xifeng 0 Mar  9 13:38 fdinfo
-rw-r--r--  1 xifeng xifeng 0 Mar  9 13:38 gid_map
-r-----  1 xifeng xifeng 0 Mar  9 13:38 io
-r--r--r--  1 xifeng xifeng 0 Mar  9 13:38 limits
-rw-r--r--  1 xifeng xifeng 0 Mar  9 13:38 loginuid
dr-x-----  2 xifeng xifeng 0 Mar  9 13:38 map_files
-r--r--r--  1 xifeng xifeng 0 Mar  9 13:38 maps
```

- `cwd` 就是当前工作目录

如何管理进程

先描述,在组织(描述进程的结构体--PCB进程控制块)

操作系统不信任任何用户---->给用户提供各种系统调用接口(接口实际上就类似于C的库函数)

进程大致了解:

- 加载到内存的程序,就叫做进程
- 任何进程在形成时,操作系统要为该进程创建PCB,进程控制块

在linux系统中, PCB---->struct task_struct{

- ```
//简单理解就是
struct task_struct{
 //进程的所有的属性数据
}
```

- 我们启动程序的过程本质都是在系统上面创建进程！！(查看进程：`ps axj | grep "test"`)(详细：`ps axj | head -1 && ps axj | grep "test"`)

The first screenshot shows a terminal window with the output of a program: "hello word!". A red box highlights the first seven lines, and a red arrow points to the text "死循环打印" (Infinite loop printing).

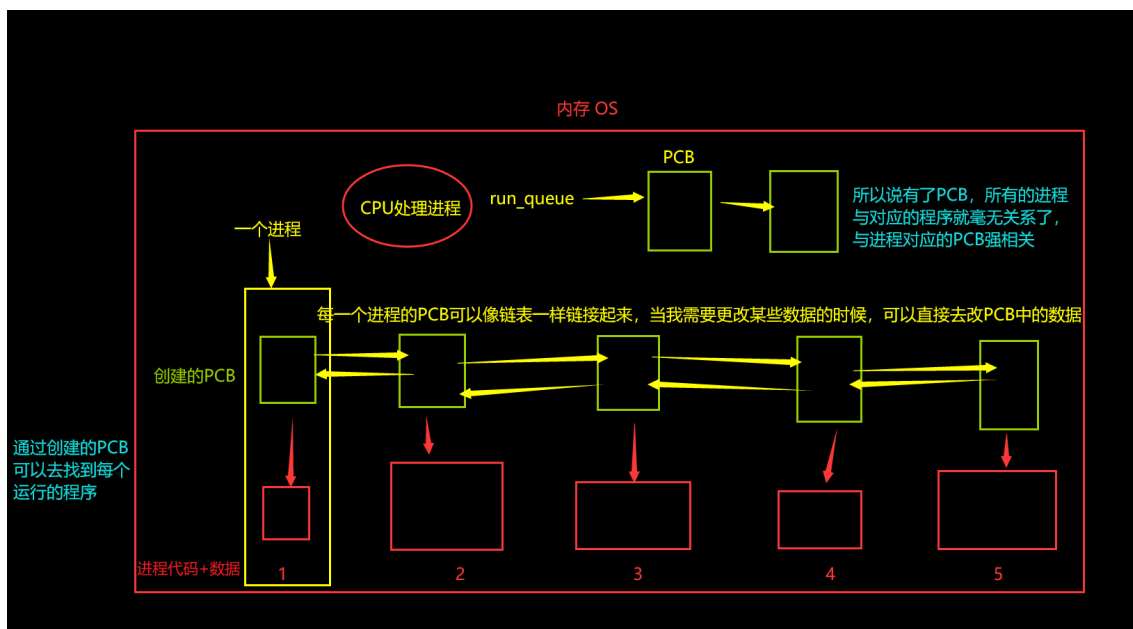
The second screenshot shows the terminal output of the command `ps axj | grep test`. A red box highlights the output, and a red arrow points to the text "退出了" (Exited).

The third screenshot shows the terminal output of the command `ps axj | head -1 && ps axj | grep "test"`. A red box highlights the output, and a red arrow points to the text "退出了" (Exited).

## 进程 vs 程序

### 进程=程序 + 操作系统维护进程相关的数据结构

- 有了进程控制块，所有的进程管理任务与进程对应的程序毫无关系！！与进程对应的内核(OS)创建的该进程的PCB强相关！！



## PCB的内部构成

- 标识符: 描述本进程的唯一标识符(PID), 用来区别其他进程
  - getpid 获取进程自身的id

```
GETPID(2) Linux Programmer's Manual

NAME
 getpid, getppid - get process identification

SYNOPSIS
 #include <sys/types.h>
 #include <unistd.h>

 pid_t getpid(void);
 pid_t getppid(void);
```

- getppid获取父进程id(在命令行上运行的命令, 基本上父进程都是bash)

```
[xifeng@VM-16-14-centos lesson8]$ ps -x | grep 9448
9447 9448 9448 9448 pts/0 17288 Ss 1001 0:00 -bash
9448 17288 17288 9448 pts/0 17288 S+ 1001 0:00 ./test
15835 18320 18319 15835 pts/1 18319 R+ 1001 0:00 grep --color=auto 9448

[xifeng@VM-16-14-centos lesson8]$
```

```
hello word! :PID ->17288 PPID->9448
hello word! :PID ->17288 PPID->9448
hello word! :PID ->17288 PPID->9448
```

- echo \$? 输出最近执行命令的退出码
- kill -9 PID就可以结束进程

```
test.c
1 #include<stdio.h>
2 #include<sys/types.h>
3 #include<unistd.h>
4 int main()
5 {
6 while(1)
7 {
8 printf("hello word! :PID ->%d\n", getpid());
9 sleep(1);
10 }
11 return 0;
12 }
```

头文件(可以man getpid)查看

获取进程的PID

```
total 20
-rw-rw-r-- 1 xifeng xifeng 60 Mar 9 10:59 Makefile
-rwxrwxr-x 1 xifeng xifeng 8464 Mar 9 11:02 test
-rw-rw-r-- 1 xifeng xifeng 163 Mar 9 11:02 test.c
[xifeng@VM-16-14-centos lesson8]$ ps axj | head -1 && ps axj | grep "test"
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
9448 15809 15809 9448 pts/0 15809 S+ 1001 0:00 ./test
15835 16096 16095 15835 pts/1 16095 R+ 1001 0:00 grep --color=
auto-test
1 19480 19480 19480 ? -1 Ssl 1001 0:00 /home/xifeng/
.VimForCpp/nvim test.c
[xifeng@VM-16-14-centos lesson8]$ kill -9 15809
[xifeng@VM-16-14-centos lesson8]$
```

```
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
hello word! :PID ->15809
Killed
[xifeng@VM-16-14-centos lesson8]$
```

- 状态
  - 任务状态, 退出代码, 退出信号等等
- 优先级: 相对于其他进程的优先级
  - 先后问题 vs 权限
    - 权限决定的是能不能
    - 优先级是已经可以只是顺序问题
- 程序计数器: 永远指向当前程序正在执行指令的下一条指令的地址
- 内存指针(可以简单理解为通过内存指针可以联系到进程的实体): 包括程序代码和进程相关数据的指针, 还有和其他进程共享的内存块的指针
- 上下文数据: 进程执行时, 所形成的处理器寄存器当中的和进程强相关的临时数据
  - 理解上下文
    1. 操作系统规定每个进程单次运行的时间片(就是一个进程单次在CPU上运行的最长时间)
    2. 在一个CPU情况下, 用户感受到的多个进程同时在运行, 本质是通过CPU的快速切换完成的

进程在运行期间是有切换的, 进程可能存在大量的临时数据-->暂时在CPU的寄存器中保存

- **保护上下文和恢复上下文**：为了让我们去做其他事情，但是不耽误当前学习，并且，当我们回来继续的时候，可以接着之前学习的内容继续学习
- **通过上下文，我们就能感受到进程是被切换的**
- **I/O 状态信息**：包括I/O请求，分配给进程的I/O设备和被进程使用的文件列表
  - OS 调度模块，较为均衡的调度(调度：获得CPU资源)每个进程
- **记账信息**：可能包括处理器时间总和，使用的时钟数总和，时间限制，记账号等

**进程=程序文件内容 +相关的数据结构**