

Linuxday12进程

1. 什么是进程？进程和程序的区别？

1. 进程是程序的一次执行过程，是一个动态的概念。
2. 程序是一个静态的文件，是指令的集合。
3. 进程是操作系统资源分配的基本单位，用PCB描述、记录进程的信息。
4. Linux中采用task_struct结构体来描述一个进程。

2. 进程标识和进程身份。

1. 进程标识：表示你是谁。

1. idle进程 系统空闲进程
2. 每个进程有一个自己标识，类型是pid_t, typedef int pid_t;
3. 每个进程都有自己的父进程，祖先进程是init进程。

2. 进程的身份：表示你能做什么（权限控制）

1. 真实用户ID和真实组ID。哪一个用户启动的进程，进程的真实用户ID就是谁。
2. 有效用户ID和有效组ID。有效用户ID和真实用户ID默认情况下是一样的。

3.

```
1 // Created Time: 2020-07-09 10:20:00
2 *****
3
4 #include <func.h> test用户以可读可写方式打开
5                               file文件会报错，权限不足
6
7 int main()
8 {
9     int fd = open("file", O_RDWR);
10    ERROR_CHECK(fd, -1, "open");
11
12    write(fd, "hello\n", 6);
13
14    close(fd);
15    return 0;
16 }
17
18 write_file.c 100% 19: 1
```

```
test@baidu: /home/ken/34th/day12$ ll
total 260
drwxrwxr-x 2 ken ken 4096 Jul 6 10:27 ./ test没有写权限
drwxrwxr-x 3 ken ken 4096 Jul 5 17:52 ../
-rw-rw-r-- 1 ken ken 0 Jul 6 10:27 file
-rwxrwxr-x 1 ken ken 8392 Jul 6 10:10 getpid*
-rw-rw-r-- 1 ken ken 427 Jul 6 09:54 getpid.c
-rwxrwxr-x 1 ken ken 8480 Jul 6 10:10 getuid*
-rw-rw-r-- 1 ken ken 384 Jul 6 10:09 getuid.c
-rw-rw-r-- 1 ken ken 119 Jul 6 09:47 Makefile
-rw-rw-r-- 1 ken ken 204236 Jul 6 09:55 pid.i
-rwxrwxr-x 1 ken ken 8432 Jul 6 10:24 write_file*
-rw-rw-r-- 1 ken ken 411 Jul 6 10:24 write_file.c
test@baidu: /home/ken/34th/day12$ ./write_file
open: Permission denied
test@baidu: /home/ken/34th/day12$ 可以执行write_file
```

4. 给可执行程序增加了s权限之后，程序执行成功。

```

[ken~/34th/day12]$ chmod u+s write_file
[ken~/34th/day12]$ ll
total 260
drwxrwxr-x 2 ken ken 4096 Jul 6 10:27 ./
drwxrwxr-x 3 ken ken 4096 Jul 5 17:52 ../
-rw-rw-r-- 1 ken ken 0 Jul 6 10:27 file
-rwxrwxr-x 1 ken ken 8392 Jul 6 10:10 getpid*
-rw-rw-r-- 1 ken ken 427 Jul 6 09:54 getpid.c
-rwxrwxr-x 1 ken ken 8480 Jul 6 10:10 getuid*
-rw-rw-r-- 1 ken ken 384 Jul 6 10:09 getuid.c
-rw-rw-r-- 1 ken ken 119 Jul 6 09:47 Makefile
-rw-rw-r-- 1 ken ken 204236 Jul 6 09:55 pid.i
-rwsrwxr-x 1 ken ken 8432 Jul 6 10:24 write_file*
-rw-rw-r-- 1 ken ken 411 Jul 6 10:24 write_file.c
[ken~/34th/day12]$ su test
Password:
test@baidu:/home/ken/34th/day12$
test@baidu:/home/ken/34th/day12$
test@baidu:/home/ken/34th/day12$ ./write_file
test@baidu:/home/ken/34th/day12$

```

5. 权限提升

1. chmod u+s

```

2. #include <func.h>
3 int main()
4 {
5     printf("uid=%d,gid=%d,euid=%d,egid=%d\n",
6           getpid(),getuid(),getgid());
7     int fd = open("file",O_RDWR);
8     ERROR_CHECK(fd,-1,"open");
9     write(fd,"hello\n",6);
10    close(fd);
11    return 0;
12 }

```

给可执行文件增加了s权限之后，程序执行时的有效用户ID就变成了程序的拥有者的ID，也就是ken

```

Total 264
drwxrwxr-x 2 ken ken 4096 Jul 6 11:09 ./
drwxrwxr-x 3 ken ken 4096 Jul 5 17:52 ../
-rw-rw-r-- 1 ken ken 6 Jul 6 11:09 file
-rwxrwxr-x 1 ken ken 8392 Jul 6 10:10 getpid*
-rw-rw-r-- 1 ken ken 427 Jul 6 09:54 getpid.c
-rwxrwxr-x 1 ken ken 8480 Jul 6 10:10 getuid*
-rw-rw-r-- 1 ken ken 384 Jul 6 10:09 getuid.c
-rw-rw-r-- 1 ken ken 119 Jul 6 09:47 Makefile
-rw-rw-r-- 1 ken ken 204236 Jul 6 09:55 pid.i
-rwsrwxr-x 1 ken ken 8656 Jul 6 11:09 write_file*
-rw-rw-r-- 1 ken ken 508 Jul 6 11:09 write_file.c
[ken~/34th/day12]$ su test
Password:
test@baidu:/home/ken/34th/day12$
test@baidu:/home/ken/34th/day12$
test@baidu:/home/ken/34th/day12$ ./write_file
uid=1001,gid=1001,euid=1000,sgid=1001
test@baidu:/home/ken/34th/day12$

```

3. 检查权限的时候，检查的是有效用户ID，有效组ID。

1. 1000 11前面两个针对可执行文件的权限 后面的这一位是对目录的权限 0 1r1w1x 111 101

chmod u+s g+s 权限提升是对可执行程序使用的。

2. 对一个可执行程序作了权限提升之后，其他用户（other）执行这个程序时，产生的进程的有效用户ID会变成可执行程序的user的ID，如果是g+s，就有了组用户的ID。

4. 粘滞位 o+t，针对的是目录。

1. 一个用户对某个目录拥有写权限，可以创建文件，也可以删除目录内的文件。

```

2. test@baidu:/home/ken/34th/day12$ cd dir/
test@baidu:/home/ken/34th/day12/dir$ touch testfile
test@baidu:/home/ken/34th/day12/dir$ ll
total 8
drwxrwxrwx 2 ken ken 4096 Jul 6 11:31 ./
drwxrwxr-x 3 ken ken 4096 Jul 6 11:28 ../
-rw-rw-r-- 1 ken ken 0 Jul 6 11:29 kenfile
-rw-rw-r-- 1 test test 0 Jul 6 11:31 testfile
test@baidu:/home/ken/34th/day12/dir$
test@baidu:/home/ken/34th/day12/dir$ test用户拥有写权限，可
test@baidu:/home/ken/34th/day12/dir$ 以删除其他用户的文件
test@baidu:/home/ken/34th/day12/dir$
test@baidu:/home/ken/34th/day12/dir$ rm kenfile
rm: remove write-protected regular empty file 'kenfile'? y
test@baidu:/home/ken/34th/day12/dir$ ll
total 8
drwxrwxrwx 2 ken ken 4096 Jul 6 11:32 ./
drwxrwxr-x 3 ken ken 4096 Jul 6 11:28 ../
-rw-rw-r-- 1 test test 0 Jul 6 11:31 testfile
test@baidu:/home/ken/34th/day12/dir$

```

3. 对rwxrwxrwx目录增加粘滞位之后，other用户可以
在该目录下创建文件，但是每个用户只能删除
属于自己的文件，不能删除其他用户的文件。

```

[ken~/34th/day12]$
[ken~/34th/day12]$ chmod o+t dir
[ken~/34th/day12]$ ll
total 268
drwxrwxr-x 3 ken ken 4096 Jul 6 11:28 ./
drwxrwxr-x 3 ken ken 4096 Jul 5 17:52 ../
drwxrwxrwt 2 ken ken 4096 Jul 6 11:32 ./
-rw-rw-r-- 1 ken ken 6 Jul 6 11:10 file
-rw-rw-r-- 1 ken ken 8202 Jul 6 10:10 notad*

```

执行效果

```

-rw-rw-r-- 1 test test 0 Jul 6 11:31 testfile
test@baidu:/home/ken/34th/day12/dir$ rm kenfile
rm: remove write-protected regular empty file 'kenfile'? y
rm: cannot remove 'kenfile': Operation not permitted
test@baidu:/home/ken/34th/day12/dir$

```

3. 进程有哪些状态

1. Linux中，常见的状态，R running 运行，S 睡眠，T 暂停，Z僵尸。
2. Linux中，R状态有两种：1. 正在CPU上运行 2. 正在准备运行（对应操作系统里的就绪状态），万事俱备，只差CPU。

```
1) TASK_RUNNING: 可运行
处于这种状态的进程，只有两种状态：
    1.1) 正在运行
        正在运行的进程就是当前进程(由current所指向的进程)
    1.2) 正准备运行
        准备运行的进程只要得到CPU就可以立即投入运行，CPU是这些进程唯一等待的系统资源，系统中有一个运行队列(run_queue)，用来容纳所有处于可运行状态的进程，调度程序执行时，从中选择一个进程投入运行。
```

4. vim

1. 安装vimplus

2. 修改文件

```
[ken~/.vim/plugged/prepare-code/snippet]$vim snippet.c
[ken~/.vim/plugged/prepare-code/snippet]$
```

5. 进程相关的命令

1. pstree 打印进程树的结构

2. echo \$\$打印当前bash的进程ID。

3. ps -elf 查看系统中的进程。ps 命令是一个采样的信息。

4. ps -aux 可以查看进程的CPU和内存的占用率。

5. top命令 动态的显示进程的信息，展示的是系统中CPU占用率最高的20个进程。

6. kill命令，给进程发信号，使用方式：kill -信号的编号 进程ID。通过kill -l可以查看所有信号。

```
OS, SIGTERM + 0, SIGTERM
[ken~/34th/day12]$kill -9 4964
[ken~/34th/day12]$
[2]- Killed ./getpid
[ken~/34th/day12]$jobs
[5]+ Running ./getpid &
[ken~/34th/day12]$kill -19 4973
[ken~/34th/day12]$

[5]+ Stopped ./getpid
[ken~/34th/day12]$
```

7. nice命令按照指定的优先级运行进程，renice命令可以修改进程的nice值。

1. nice -n 可执行程序

2. renice -n 指定的nice值 -p 进程ID

6. 进程的优先级

1. 范围 0-139, 0-99表示的是动态优先级, priority, 100-139 静态优先级, nice值。

2. 调度策略:

1. 实时调度策略:

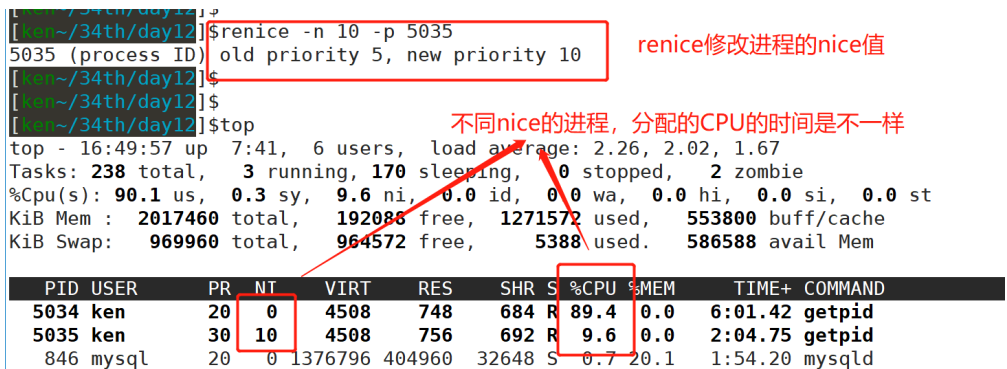
1. SCHED_FIFO策略: 相同优先级的进程, 先来先服务, 知道先来的进程执行完毕或者主动让出CPU, 后面来的**同样优先级**进程才能得到执行。

2. SCHED_RR (round robin)策略, 轮转的方式, 相同优先级的进程, 会轮流分配CPU。

3. 实时的进程优先级更高, 能够抢占优先级低的进程。

4. 相同优先级的实时进程, 具体调度就根据 SCHED_FIFO和SCHED_RR策略调度。

2. 非实时 (普通调度策略): CFS 完全公平调度策略, nice 值, top命令看到的nice值的范围-20到19, nice值越大, 优先级越低, 分配的时间片越少。

3.  renice修改进程的nice值

不同nice的进程, 分配的CPU的时间是不一样的

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5034	ken	20	0	4508	748	684	R	89.4	0.0	6:01.42	getpid
5035	ken	30	10	4508	756	692	R	9.6	0.0	2:04.75	getpid
846	mysql	20	0	1376796	404960	32648	S	0.7	20.1	1:54.20	mysqld

7. 会话, 进程组, 前台进程和后台进程

1. 会话, 控制终端, 会话中的首进程是bash进程, 一个会话下面可以有多个进程组。包括一个前台进程组和若干个后台进程组。

2. 前台进程组, 可以接受控制终端上传输的数据。

3. 后台运行一个进程, 在执行程序时后面加一个&符号, 变成了后台进程。

```

[ken~/34th/day12]$
[ken~/34th/day12]$ ./getpid
pid = 4797,ppid=1974
hello
^C
[ken~/34th/day12]$
[ken~/34th/day12]$
[ken~/34th/day12]$ ./getpid &
[1] 4801
[ken~/34th/day12]$ pid = 4801,ppid=1974

```

4. 通过jobs命令可以看到当前会话下面的后台作业，每个作业都有一个编号。可以通过fg+作业编号把后台运行的作业拉回到前台。拉回到前台之后，就可以通过控制终端跟前台进程交互。

```

[ken~/34th/day12]$
[ken~/34th/day12]$ jobs
[1]+  Running                  ./getpid &
[ken~/34th/day12]$ fg 1
./getpid
hello
^C
[ken~/34th/day12]$ ps -elf|qrep getpid

```

ctrl+z把当前的前台进程挂起，到后台，变成T状态。

```
[ken~/34th/day12]$ ./getpid
pid = 4955, ppid=1974
^Z
[1]+  Stopped                  ./getpid
[ken~/34th/day12]$ bg 1
[1]+  ./getpid &
[ken~/34th/day12]$ jobs
[1]+  Running                  ./getpid &
[ken~/34th/day12]$ fg 1
./getpid
^Z
[1]+  Stopped                  ./getpid
[ken~/34th/day12]$ bg 1
[1]+  ./getpid &
[ken~/34th/day12]$ fg 1
./getpid
^C
[ken~/34th/day12]$
```

发信号把进程挂起

bg+作业编号, 放到后台运行, 进程从T状态变成R状态

fg 命令拉回到前台