
LINUX进程相关

目录

第一章 linux进程相关

第一节 查看进程

第二节 创建进程

1 c / 2 linux脚本 /

3 父子进程 /

第三节 无限创建进程

第四节 问题

1 一、 / 2 二、 /

参考网址：

developer

掘金

主讲人：刘锦乐

查看进程

先来看看linux下怎么查看进程：

```
ps -ef | grep {programname}
```

其中ps是 **process status** 的缩写,用于查看进程， **| grep** 用于连接，上面的命令用于查询指定程序的相关进程，如查询firefox：

```
ps -ef | grep firefox
heeler 13439 1612 10 15:10 ? 00:00:44 /usr/lib/firefox/firefox
heeler 13529 13439 0 15:10 ? 00:00:00 /usr/lib/firefox/firefox -contentproc -parentBuildID 20220304162637 -prefsLen 1 -prefMapSize 252961 -appDir /usr/lib/firefox/browser 13439 true socket
heeler 13560 13439 0 15:10 ? 00:00:01 /usr/lib/firefox/firefox -contentproc -childID 1 -isForBrowser -prefsLen 139 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 13629 13439 2 15:10 ? 00:00:09 /usr/lib/firefox/firefox -contentproc -childID 2 -isForBrowser -prefsLen 4849 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 13693 13439 2 15:10 ? 00:00:10 /usr/lib/firefox/firefox -contentproc -childID 3 -isForBrowser -prefsLen 5484 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 13700 13439 1 15:10 ? 00:00:06 /usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 5484 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 13709 13439 0 15:10 ? 00:00:02 /usr/lib/firefox/firefox -contentproc -childID 5 -isForBrowser -prefsLen 5484 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 13975 13439 1 15:11 ? 00:00:04 /usr/lib/firefox/firefox -contentproc -childID 6 -isForBrowser -prefsLen 5611 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 14550 13439 0 15:11 ? 00:00:00 /usr/lib/firefox/firefox -contentproc -childID 7 -isForBrowser -prefsLen 5768 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 21281 13439 0 15:15 ? 00:00:00 /usr/lib/firefox/firefox -contentproc -parentBuildID 20220304162637 -prefsLen 5816 -prefMapSize 252961 -appDir /usr/lib/firefox/browser 13439 true rdd
heeler 21283 13439 0 15:15 ? 00:00:00 /usr/lib/firefox/firefox -contentproc -childID 8 -isForBrowser -prefsLen 5816 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 21939 13439 0 15:16 ? 00:00:00 /usr/lib/firefox/firefox -contentproc -childID 9 -isForBrowser -prefsLen 5816 -prefMapSize 252961 -jsInitLen 279340 -parentBuildID 20220304162637 -appDir /usr/lib/firefox/browser 13439 true tab
heeler 24669 23797 0 15:18 pts/1 00:00:00 grep --color=auto --exclude-dir=.bzz --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn --exclude-dir=.idea --exclude-dir=.tox firefox
```

上图截取一部分

```
> ps -ef | grep firefox
heeler      13439      1612   8 15:10 ?          00:00:48 /usr/lib/firefox/firefox
```

其中信息按照下面的方式排列：



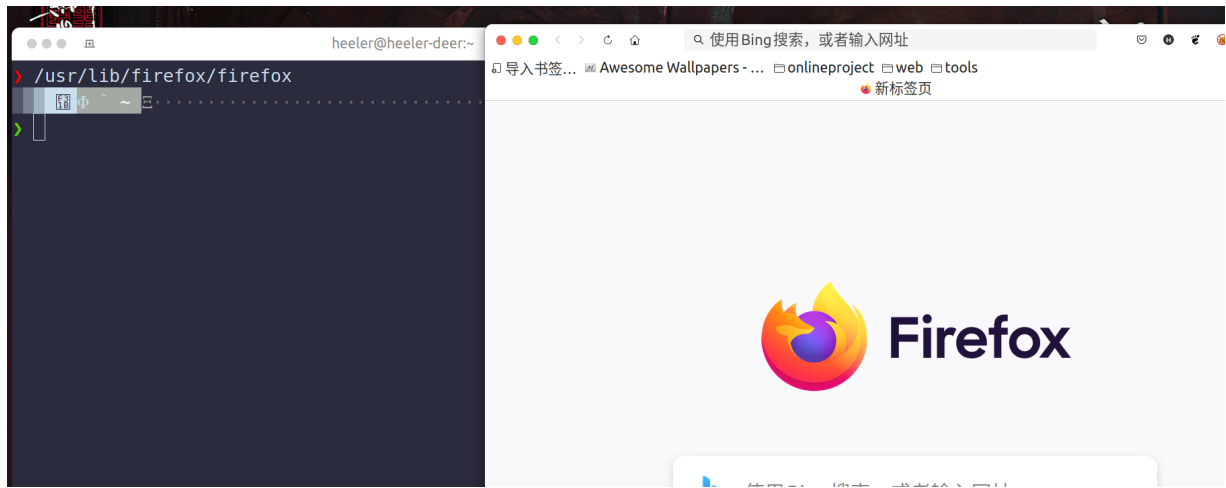
UID	PID	PPID	C	STIME	TTY	TIME	CMD
-----	-----	------	---	-------	-----	------	-----

即进程创建的用户id,进程的pid,该进程的父进程的id，c指cpu占用率，stime是进程启动时间，tty代表终端（？即本地终端），time指进程占用cpu的总时间，cmd就是command,在linux中就是这样的路径显示,比如我输入



```
/usr/lib/firefox/firefox
```

就会启动firefox



创建进程

H3 C

下面以c为例子，讲解如何创建进程.

```
void test_fork_1()
{
    fork();
    fork();
    fork();
    // "hello"的打印次数等于创建的进程数, 进程总数为
    2^n, 其中n是fork调用的数目
    fprintf(stdout, "hello\n"); // 注: 总共会输出
    8次hello
}
```

由于fork会从其所在的行创建一个子进程，所以上面代码会创建总共 $2^n = 8$ 个子进程。

我们用fork函数写一个 1.c 文件：

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main( ){
    pid_t child_pid;
    child_pid = fork (); // Create a new child
    process;
    if (child_pid < 0) {
        printf("fork failed");
    }
}
```

```

        return 1;
    } else if (child_pid == 0) {
        printf ("child process successfully
created!\n");
        printf ("child_PID = %d,parent_PID =
%d\n",
            getpid(), getppid( ) );
    } else {
        wait(NULL);
        printf ("parent process successfully
created!\n");
        printf ("child_PID = %d, parent_PID =
%d", getpid( ), getppid( ) );
    }
    return 0;
}

```

结果：

```

> gcc -g -o1 1.c
1.c: In function 'main':
1.c:15:7: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
   15 |     wait(NULL);
      |     ^~~~
> ./1
child process successfully created!
child_PID = 54877,parent_PID = 54876
parent process successfully created!
child PID = 54876, parent PID = 50772%

```

H3 linux脚本

下面展示linux下在父进程被杀死的情况下子进程仍可以运行的方法：

先写一个 2.sh 脚本：

```
while sleep 30; do date;done
```

以该脚本为例，执行两次，

```
nohup sh 2.sh&  
nohup zsh 2.sh&
```

可以看到两个进程的父进程都是 85048

heeler	86132	85048	0	15:58	pts/1	00:00:00	sh 2.sh
heeler	86133	86132	0	15:58	pts/1	00:00:00	sleep 30
heeler	86270	85048	0	15:58	pts/1	00:00:00	zsh 2.sh


```
heeler      85048    50301    0 15:58 pts/1      00:00:00 /usr/bin/zsh
```

下面我们杀掉85048进程：

```
75944    45325    0 15:51 ?          00:00:00 /opt/google/chrome/chrome --type=re
81661         2    0 15:55 ?          00:00:00 [kworker/1:0-events]
81662         2    0 15:55 ?          00:00:00 [kworker/9:2-events]
84049         2    0 15:57 ?          00:00:00 [kworker/14:0-events]
84050         2    0 15:57 ?          00:00:00 [kworker/14:3-events]
84156         2    0 15:57 ?          00:00:00 [kworker/u33:0-rtw89_tx_wq]
85121         2    0 15:58 ?          00:00:00 [kworker/15:1-mm_percpu_wq]
85267         2    0 15:58 ?          00:00:00 [kworker/u32:1-events_unbound]
86132    1612    0 15:58 ?          00:00:00 sh 2.sh
86270    1612    0 15:58 ?          00:00:00 zsh 2.sh
86387    1612    0 15:58 ?          00:00:00 /usr/bin/zsh
```

22	2022年	03月	08日	星期一	15:52:27	CST
23	2022年	03月	08日	星期二	15:52:42	CST
24	2022年	03月	08日	星期二	15:52:57	CST
25	2022年	03月	08日	星期二	15:53:12	CST
26	2022年	03月	08日	星期二	15:53:27	CST
27	2022年	03月	08日	星期二	15:53:42	CST
28	2022年	03月	08日	星期二	15:53:57	CST
29	2022年	03月	08日	星期二	15:54:12	CST
30	2022年	03月	08日	星期二	15:54:27	CST
31	2022年	03月	08日	星期二	15:54:42	CST
32	2022年	03月	08日	星期二	15:59:22	CST
33	2022年	03月	08日	星期二	15:59:27	CST
34	2022年	03月	08日	星期二	15:59:52	CST
35	2022年	03月	08日	星期二	15:59:57	CST
36	2022年	03月	08日	星期二	16:00:22	CST
37	2022年	03月	08日	星期二	16:00:27	CST
38	2022年	03月	08日	星期二	16:00:52	CST
39	2022年	03月	08日	星期二	16:00:57	CST
40	2022年	03月	08日	星期二	16:01:22	CST
41	2022年	03月	08日	星期二	16:01:27	CST
42						

可以看到子进程仍在执行

H3 父子进程

以 3.c 为例子，讲解linux对于简单情况下父子进程中虚拟地址空间的处理：



```
#include <unistd.h>
#include <stdio.h>

int main(){
    // 创建进程
    pid_t pid = fork();

    // 局部变量
    int num = 10;

    // 判断当前进程是父进程 还是子进程
    if (pid > 0){                // 进程号 > 0, 即为
子进程的进程号, 当前为父进程
        printf("I am parent process, pid: %d,
ppid: %d\n", getpid(), getppid());

        printf("parent process num : %d\n",
num);
        num += 10;
        printf("parent process num + 10 :
%d\n", num);
    }
    else if (pid == 0){        // 进程号 == 0, 表示
当前为子进程
        printf("I am child process, pid: %d,
ppid: %d\n", getpid(), getppid());

        printf("child process num : %d\n",
num);
    }
}
```

```
        num += 100;
        printf("child process num + 100 :
%d\n", num);
    }

    return 0;
}
```

代码执行结果：

```
> gcc -o 3 3.c
> ./3
I am parent process, pid: 97839, ppid: 97597
parent process num : 10
parent process num + 10 : 20
I am child process, pid: 97840, ppid: 97839
child process num : 10
child process num + 100 : 110
```

linux采取的策略是 **读时共享，写时拷贝**，即仅在对子进程执行写操作时才复制父进程的内容到子进程，否则只是pid不同（对于fork来讲）

无限创建进程

由于fork方便创建进程，我们仍以c中的fork为例，尝试用无限循环创建进程：

在 4.c 中，

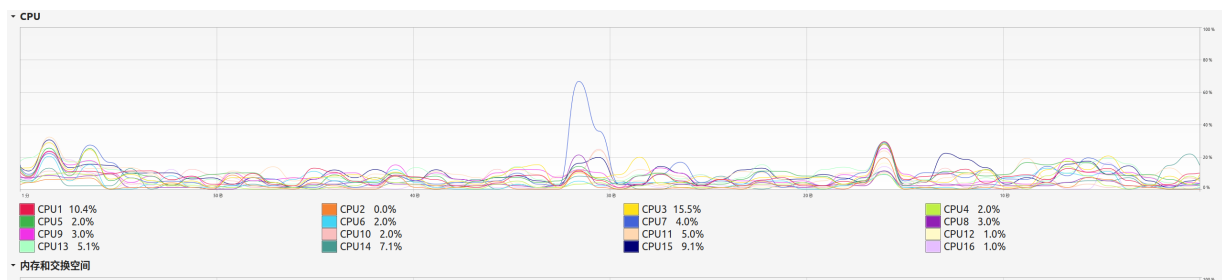
```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main( ){
    pid_t child_pid;
    while(1){
        child_pid = fork (); // Create a new child
process;
        if (child_pid < 0) {
            printf("fork failed");
            return 1;
        } else if (child_pid == 0) {
            printf ("child process successfully
created!\n");
            printf ("child_PID = %d,parent_PID =
%d\n",
                getpid(), getppid( ) );
        } else {
            wait(NULL);
        }
    }
}
```

```

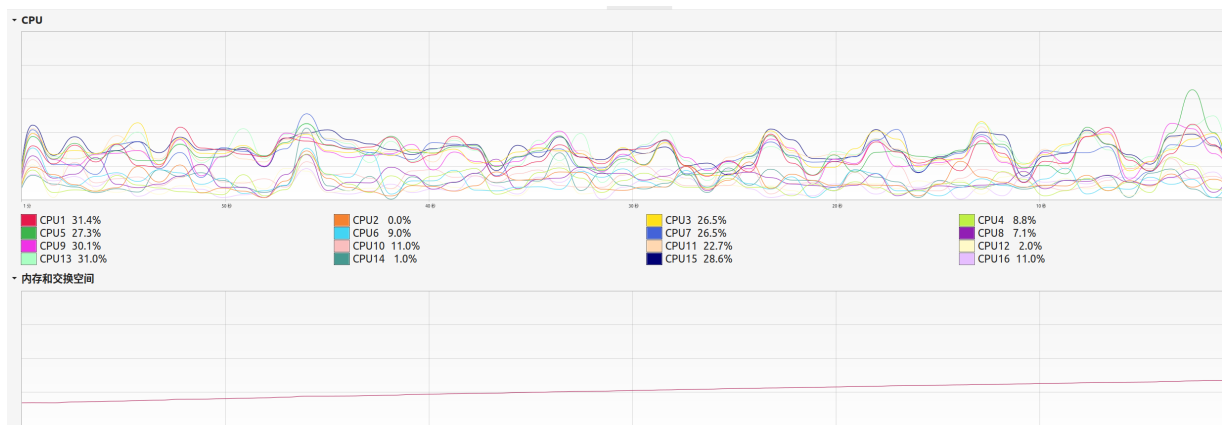
    printf ("parent process successfully
created!\n");
    printf ("child_PID = %d, parent_PID =
%d", getpid( ), getppid( ) );
    }
    }
    return 0;
}

```

正常执行cpu状况：



无限循环结果：



可以看到运行几十秒后我的cpu占用以及有了明显的变化。而由于CPU的寻址空间是有限的，所以不能无限创进程。

问题

H3 一、

进程是什么？为什么在终端 `pkill firefox` 后火狐浏览器就自己关闭了？

H3 二、

如果cpu等硬件资源无限，能否创建无限多的进程？

