更多嵌入式 Linux 学习资料，请关注：一口 Linux 回复关键字:1024

个人微信

公众号

一、粉丝提问

fork 出的进程的父进程是从哪来的？

粉丝提问，一口君必须满足

二、解答

这个问题看上去很简单，但是要想把进程的父进程相关的所有知识点搞清楚，还是有点难度的，下面我们稍微拓展下，分几点来讲解这个知识点。

1. 如何查看进程 ID

每个 linux 进程都一定有一个唯一的数字标识符，称为进程 ID（process ID），进程 ID 总是一非负整数，它的父进程叫 PPID。

查看进程 ID 命令：

```
ps -ef
```

```
peng@ubuntu:~$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 06:43 ?        00:00:00 /sbin/init
root           2     0  0 06:43 ?        00:00:00 [kthreadd]
root           3     2  0 06:43 ?        00:00:00 [ksoftirqd/0]
root           4     2  0 06:43 ?        00:00:00 [kworker/0:0]
root           6     2  0 06:43 ?        00:00:00 [migration/0]
root           7     2  0 06:43 ?        00:00:00 [watchdog/0]
root           8     2  0 06:43 ?        00:00:00 [cpuset]
root           9     2  0 06:43 ?        00:00:00 [khelper]
root          10     2  0 06:43 ?        00:00:00 [kdevtmpfs]
root          11     2  0 06:43 ?        00:00:00 [netns]
```

进程ID　父进程ID　进程名

查看进程

也可以使用函数来获得进程 ID：

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);    返回：调用进程的进程 ID
pid_t getppid(void);    返回：调用进程的父进程 ID
```

2. 第一个进程 init

Linux 内核启动之后，会创建第一个用户级进程 init，由上图可知，init 进程（pid=1）是除了 idle 进程（pid=0，也就是 init_task）之外另一个比较特殊的进程，它是 Linux 内核开始建立起进程概念时第一个通过 kernel_thread 产生的进程，其开始在内核态执行，然后通过一个系统调用，开始执行用户空间的 /sbin/init 程序。

3. fork 函数

创建一个进程很简单，先来认识一下 fork 函数：

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);

返回：子进程中为0，父进程中为子进程 I D，出错为-1
```

由 fork 创建的新进程被称为子进程（ child process）。该函数被调用一次，但返回两次，两次返回的区别是子进程的返回值是 0，而父进程的返回值则是子进程的进程 ID。

一般来说，在 f o r k 之后是父进程先执行还是子进程先执行是不确定的。这取决于内核所使用的调度算法。

「举例：」

```
#include <unistd.h>

int main()
{
 pid_t pid;

 if ((pid = fork()) == -1) {
  perror("fork");
  return  -1;
 } else if (pid == 0) {
  /* this is child process */
  printf("The return value is %d  In child process!!  My PID is %d,  My PPID is %d\
n", pid,getpid(), getppid());

 } else {
  /* this is parent */
  printf("The return value is %d In parent process!!  My PID is %d,  My PPID is %d\
n", pid,getpid(), getppid());
 }
 return  0;
}
```

执行结果：

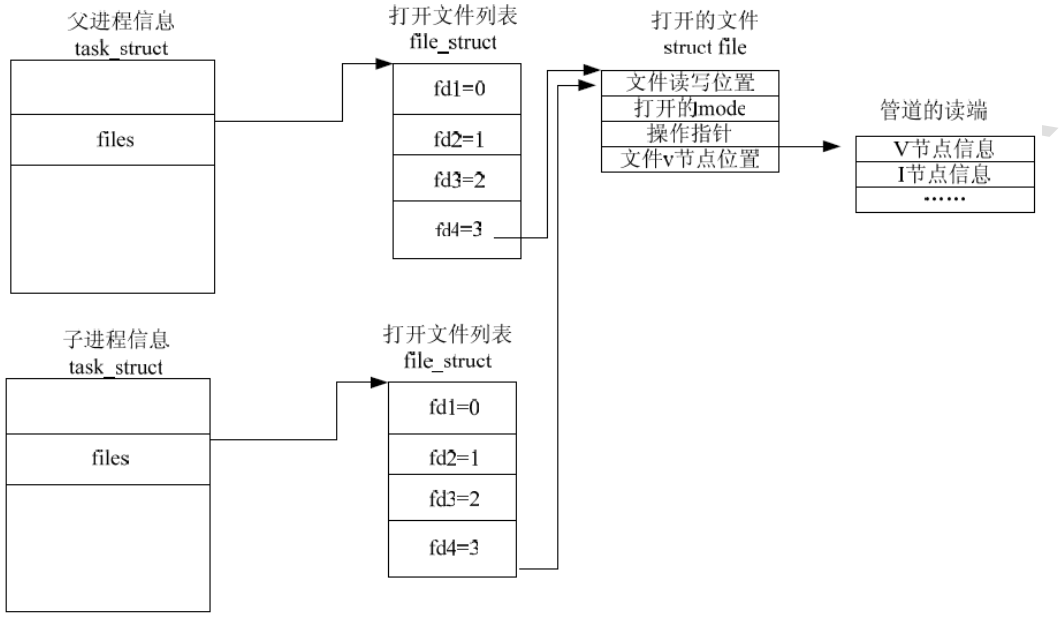
```
peng@ubuntu:~/test$ ./a.out
The return value is 3186 In parent process!!  My PID is 3185,  My PPID is 2675
peng@ubuntu:~/test$ The return value is 0  In child process!!  My PID is 3186,  My PPID is 1
```

fork

由上图可知，fork 被调用了一次，返回了两次。

【拓展】

使用 fork 函数得到的子进程是父进程的处继承了整个进程的地址空间，包括：**「进程上下文、进程堆栈、内存信息、打开的文件描述符、信号控制设置、进程优先级、进程组号、当前工作目录、根目录、资源限制、控制终端」**等。



fork

fork 出的子进程会集成父进程的文件描述符，如果读写文件，父子进程之间会互相影响。

4. ./run 运行的程序父进程是谁？

我们来编写一个例子：

```
int main(int argc, const char *argv[])
{
        while(1);
        return 0;
}
```

编译执行

```
gcc test.c
./a.out
```

例子很简单，就是创建一个死循环的进程。

```
ps -ef 查看进程 ID
```

```
peng      2672  2665  0 19:20 ?        00:00:00 gnome-pty-helper
peng      2675  2665  0 19:20 pts/2    00:00:00 bash
peng      2712     1  0 19:20 ?        00:00:00 /usr/lib/unity-lens-music/unity-musicstore
peng      2748     1  0 19:20 ?        00:00:00 /usr/bin/python /usr/lib/unity-scope-video
peng      2766  2281  0 19:21 ?        00:00:00 update-notifier
root      2779     1  0 19:21 ?        00:00:00 /usr/bin/python /usr/lib/system-service/sy
peng      2797  2281  0 19:22 ?        00:00:00 /usr/lib/deja-dup/deja-dup/deja-dup-monito
peng      2923  2665  0 19:41 pts/3    00:00:00 bash
peng      2991  2675 97 19:42 pts/2    00:00:13 ./a.out
peng      2996  2923  0 19:42 pts/3    00:00:00 ps -ef
```

执行结果

由上图可知，a.out 进程的进程 ID 是 2991，父进程 ID 是 2675，即进程 bash：

```
peng      2665  2664  0 19:20 ?        00:00:00 gnome-terminal
```

2665

bash 的父进程是 gnome-terminal，所以大家应该明白，我们打开 1 个 Linux 终端，其实就是启动了 1 个 gnome-terminal 进程。我们在这个终端上执行./a.out 其实就是利用 gnome-terminal 的子进程 bash 通过 execve()将创建的子进程装入 a.out：

```
peng@ubuntu:~/test$ strace ./a.out
execve("./a.out", ["./a.out"], [/* 51 vars */]) = 0
brk(0)                                  = 0x8337000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7700000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=99567, ...}) = 0
mmap2(NULL, 99567, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76e7000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\0004\0\0\0"..., 512) =
fstat64(3, {st_mode=S_IFREG|0755, st_size=1713640, ...}) = 0
mmap2(NULL, 1723100, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7542000
mmap2(0xb76e1000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0
mmap2(0xb76e4000, 10972, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7541000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7541900, limit:1048575, seg_32bit:1,
ent:0, useable:1}) = 0
mprotect(0xb76e1000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7723000, 4096, PROT_READ)   = 0
munmap(0xb76e7000, 99567)               = 0
```

strace

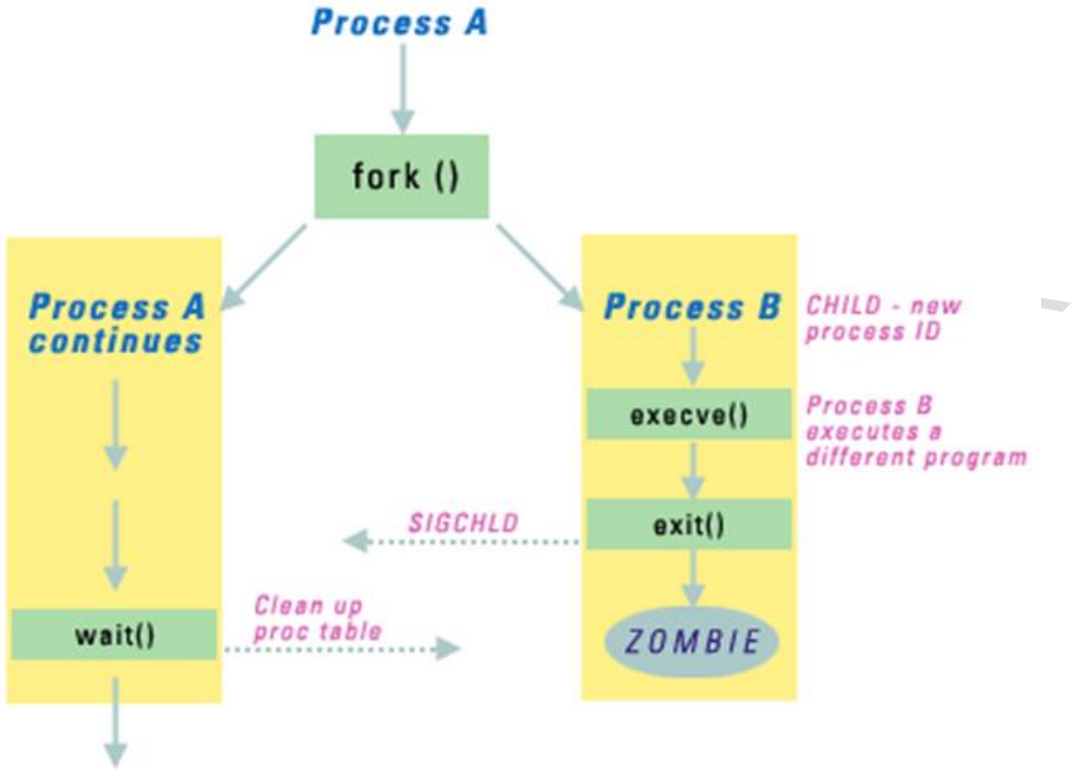
5. 进程和终端的关系

关于进程和终端的关系可以参考以下文章：

《进程组、会话、控制终端概念，如何创建守护进程？》

6. 父进程死了，子进程怎么办？

1）僵尸进程



僵尸进程

如上图所示，

1. 父进程 Process A 创建子进程 Process B，当子进程退出时会给父进程发送信号 SIGCHLD;
2. 如果父进程没有调用 wait 等待子进程结束，退出状态丢失，转换成僵死状态，子进程会变成一个僵尸进程。

当父进程调用 wait，僵尸子进程的结束状态被提取出来，子进程被删除，并且 wait 函数立刻返回。

实例

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

/* create a ZOMBIE
 * ps -ax | grep a.out to show the zombie
 */
int main()
{
 if(fork()) {
```

```
  //父进程
  while(1){
    sleep(1);
  }
 }
 //子进程
}
```

上述程序会保证父进程不退出，一直在 while(1)中无限循环，而子进程会立刻退出。

```
root@ubuntu:/home/peng/test# ps -ef | grep a.out
peng      3096  2675  0 20:19 pts/2    00:00:00 ./a.out
peng      3097  3096  0 20:19 pts/2    00:00:00 [a.out] <defunct>
root      3108  3031  0 20:21 pts/3    00:00:00 grep --color=auto a.out
```

僵尸进程

由上图可知，父进程是 3096，子进程是 3097，子进程因为退出后父进程没有调用 wait 回收子进程资源，所以子进程 3097 变成僵尸进程 defunct。

```
ps -ax  | grep a.out 查看进程状态
```

```
root@ubuntu:/home/peng/test# ps -ax | grep a.out
 3096 pts/2    S+     0:00 ./a.out
 3097 pts/2    Z+     0:00 [a.out] <defunct>
 3114 pts/3    S+     0:00 grep --color=auto a.out
```

僵尸进程

2）孤儿进程

如果父进程退出，并且没有调用 wait 函数，它的子进程就变成孤儿进程，会被一个特殊进程继承，这就是 init 进程，init 进程会自动清理所有它继承的僵尸进程。

实例代码：

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

int main()
{
 if(fork()) {
  //父进程

 }else{
```

```
    //子进程
    while(1){
      sleep(1);
    }
  }
}
```

上述程序会保证子进程不退出，一直在 while(1)中无限循环，而父进程会立刻退出。

孤儿进程：

```
root@ubuntu:/home/peng/test# ./a.out
root@ubuntu:/home/peng/test# ps -ef | grep a.out
root       3152      1  0 20:45 pts/3    00:00:00 ./a.out
root       3154   3031  0 20:45 pts/3    00:00:00 grep --color=auto a.out
```

孤儿进程

./a.out 的父进程 ID 变成 1，所以该子进程被 init 进程继承。

三、其他启动进程的方法

1. exec 族函数

1. fork 函数用于创建一个子进程，该子进程几乎拷贝了父进程的全部内容。
2. exec 函数族提供了一种在进程中启动另一个程序执行的方法。它可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段。在执行完之后，原调用进程的内容除了进程号外，其他全部都被替换了。
3. 可执行文件既可以是二进制文件，也可以是任何 Linux 下可执行的脚本文件。

每当进程调用一种 exec 函数时，该进程完全由新程序代换，而新程序从 main 函数开始执行。Exec 并不创建新进程，所以前后进程 ID 也不会变。Exec 只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。

「何时使用？」

1. 当进程认为自己不能再为系统和用户做出任何贡献了时就可以调用 exec 函数，让自己执行新的程序
2. 如果某个进程想同时执行另一个程序，它就可以调用 fork 函数创建子进程，然后在子进程中调用任何一个 exec 函数。这样看起来就好像通过执行应用程序而产生了一个新进程一样。

「函数原型」

所需头文件	#include <unistd.h>
函数原型	int execl(const char *path, const char *arg, ...);
	int execv(const char *path, char *const argv[]);
	int execle(const char *path, const char *arg, ..., char *const envp[]);
	int execve(const char *path, char *const argv[], char *const envp[]);
	int execlp(const char *file, const char *arg, ...);
	int execvp(const char *file, char *const argv[]);
函数返回值	-1：出错

函数原型

2. cron 命令

在 Linux 系统中，计划任务一般是由 cron 承担，我们可以把 cron 设置为开机时自动启动。cron 启动后，它会读取它的所有配置文件（全局性配置文件/etc/crontab，以及每个用户的计划任务配置文件），然后 cron 会根据命令和执行时间来按时来调用度工作任务。

检查 cron 是否安装：

```
ps -ef | grep cron
```

```
peng@ubuntu:~/test$ ps -ef | grep cron
root      1096     1  0 19:15 ?        00:00:00 cron
peng      3164  2675  0 20:56 pts/2    00:00:00 grep --color=auto cron
```

cron

```
crontab -u //设定某个用户的cron服务，一般root用户在执行这个命令的时候需要此参数
crontab -l //列出某个用户cron服务的详细内容
crontab -r //删除某个用户的cron服务
crontab -e //编辑某个用户的cron服务
```

root 查看自己的 cron 设置：

```
crontab -u root -l
```

或者直接看自己名下的任务：

```
 crontab -l
```

创建任务：

```
crontab -e
```

打开默认编辑器编辑后保存退出即可

编辑基本格式 ：

```
*****command
分 时 日 月 周 命令
第 1 列表示分钟 1～59 每分钟用*或者 */1 表示
第 2 列表示小时 1～23（0 表示 0 点）
第 3 列表示日期 1～31
第 4 列表示月份 1～12
第 5 列标识号星期 0～6（0 表示星期天）
第 6 列要运行的命令
如果写为*， 表示每 X
如果想定义间隔，在 X 后加"/"和间隔的数字
```

每隔一分钟打印一下系统时间

```
*/1 * * * * date >> ~/t.log   //>> means append
```

3. at

在 linux 系统如果你想要让自己设计的备份程序可以自动在某个时间点开始在系统底下运行，而不需要手动来启动它，又该如何处置呢？

这些例行的工作可能又分为一次性定时工作与循环定时工作，在系统内又是哪些服务在负责？

还有，如果你想要每年在老婆的生日前一天就发出一封信件提醒自己不要忘记，linux 系统下该怎么做呢？

但是 crontab 主要用来提交不断循环执行的 job， 而 at 用来提交一段时间后执行的 job（执行完就自动删除整个 job）

「举例：」

1） 首先检查 atd 服务有无开启 在一个指定的时间执行一个指定任务，只能执行一次，且需要开启 atd 进程

```
ps -ef | grep atd 查看，
开启用/etc/init.d/atd start or restart;
开机即启动则需要运行 chkconfig --level 2345 atd on
```

2） 定时在 11:30am 用 ls 列出当前目录内容并写入~/log 文件

```
cd ~
at 11:30am today
```

```
at>ls > ~/t.log
at> <EOT> //按Ctl-D 退出
```