**进程笔记**

目录

[要点： 2](#_Toc486427281)

[相关函数： 3](#_Toc486427282)

[创建子进程fork 3](#_Toc486427283)

[执行进程execl 4](#_Toc486427284)

[等待清理进程waitpid 5](#_Toc486427285)

[获取pid/ppid getpid()/getppid() 6](#_Toc486427286)

# 要点：

1.进程相当于自我复制，将自己复制到另一个内存区域执行。复制出来的进程叫做子进程，进程之间相互独立（全局变量也是独立拥有的）。

2. Fork创建的新进程是和父进程（除了PID和PPID）一样的副本，包括真实和有效的UID和GID、进程组合会话ID、环境、资源限制、打开的文件以及共享内存段。

3. 如果fork执行成功，就向父进程返回子进程的PID，并向子进程返回0。这就一起这即使你只调用fork一次，他也会返回两次。

4.子进程与父进程没有绝对的先后执行顺序。

5.子进程死后就会变成僵尸进程，需要父进程处理尸体（waitpid）。否则子进程要等到父进程结束才会退还资源。

# 相关函数：

## 创建子进程fork

|  |  |
| --- | --- |
| **pid\_t fork(void)** | |
| void | None |
|  | |
| 头文件 | #include<unistd.h> |
| 返回值 | 向父进程返回子进程的PID，并向子进程返回0。错误返回-1。 |
| 说明：  创建一个子进程。 | |
| 示例：  pid\_t PID;  PID = fork(); | |

## 执行进程execl

|  |  |
| --- | --- |
| **int execl(const char \*path, const char\*arg, ...)** | |
| path | 要执行的二进制文件或脚本的完整路径 |
| arg | 是要传递给程序的完整参数 |
|  | |
| 头文件 | #include<unistd.h> |
| 返回值 | 若是成功建立进程返回0,否则返回-1 |
| 说明：  这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源被收回 | |
| 示例：  execl("/bin/ls", "ls","-al","/etc/passwd",(char\*)0); | |

## 等待清理进程waitpid

|  |  |
| --- | --- |
| **pid\_t waitpid(pid\_t pid,int \* status,int options)** | |
| pid | 进程的PID  **pid>0:**  只等待进程ID等于pid的子进程，不管其它已经有多少子进程运行结束退出了，只要指定的子进程还没有结束，waitpid就会一直等下去。  **pid=-1:**  等待任何一个子进程退出，没有任何限制，此时waitpid和wait的作用一模一样。 |
| **pid=0:**  等待同一个进程组中的任何子进程，如果子进程已经加入了别的进程组，waitpid不会对它做任何理睬。  **pid<-1:**  等待一个指定进程组中的任何子进程，这个进程组的ID等于pid的绝对值。 |
| status | **传递出进程结束时的状态。不在意可使用NULL** |
| options | **WNOHANG:**  即使没有子进程退出，它也会立即返回，不会像wait那样永远等下去。 |
|  | |
| 头文件 | #include<sys/wait.h> |
| 返回值 | 函数若成功，返回进程PID，若出错则返回-1； |
| 说明：  大部分情况，我们只是要清理尸体而已。  用waitpid(-1, NULL,0); 或者wait(NULL); | |
| 示例：  waitpid(-1, NULL,0); | |

## 获取pid/ppid getpid()/getppid()

|  |  |
| --- | --- |
| **pid\_t getpid()/getppid()** | |
| void | None |
|  | |
| 头文件 | #include<unistd.h> |
| 返回值 | 返回该进程的PID/返回父进程PID（即PPID）。错误返回：-1。 |
| 说明：  获取PID。 | |
| 示例：  pid\_t pid;  pid = getpid (); | |