

# CS302 OS Lab04 - Report

Name: 刘仁杰

SID: 11911808

## Answers

### 1.代码中如何区分父子进程？父子进程的执行顺序是否是固定的？

- 如何区分父子进程：

父进程 `fork()` 返回值是子进程pid，而子进程 `fork()` 返回值是0。

- 父子进程的执行顺序：

父子进程的执行顺序不固定，由OS来调度。

### 2.waitpid()函数的原型是什么，参数的含义是什么，具体功能是什么？

- 函数原型：

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options)
```

- 参数含义：

pid：

1. pid>0时，只等待进程ID等于pid的子进程。
2. pid=-1时，等待任何一个子进程，此时waitpid和wait的作用一模一样。
3. pid=0时，等待同一个进程组中的任何子进程。
4. pid<-1时，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：

如果 `status` 不为 `NULL`，那么 `waitpid()` 将会把子进程结束返回的状态信息作为一个整形值存到传进来的指针。

options：一些额外的选项来控制waitpid，支持三个选项，可以用 `or` 运算符把它们连接起来使用

1. `WNOHANG`：即使没有子进程退出，它也会立即返回。
2. `WUNTRACED`：同时也会返回子进程是否停止的信息，以及停止的子进程的状态信息。
3. `WCONTINUED`：同时也会返回一个停止的子进程是否通过 `SIGCONT` 被恢复运行了。

- 具体功能：

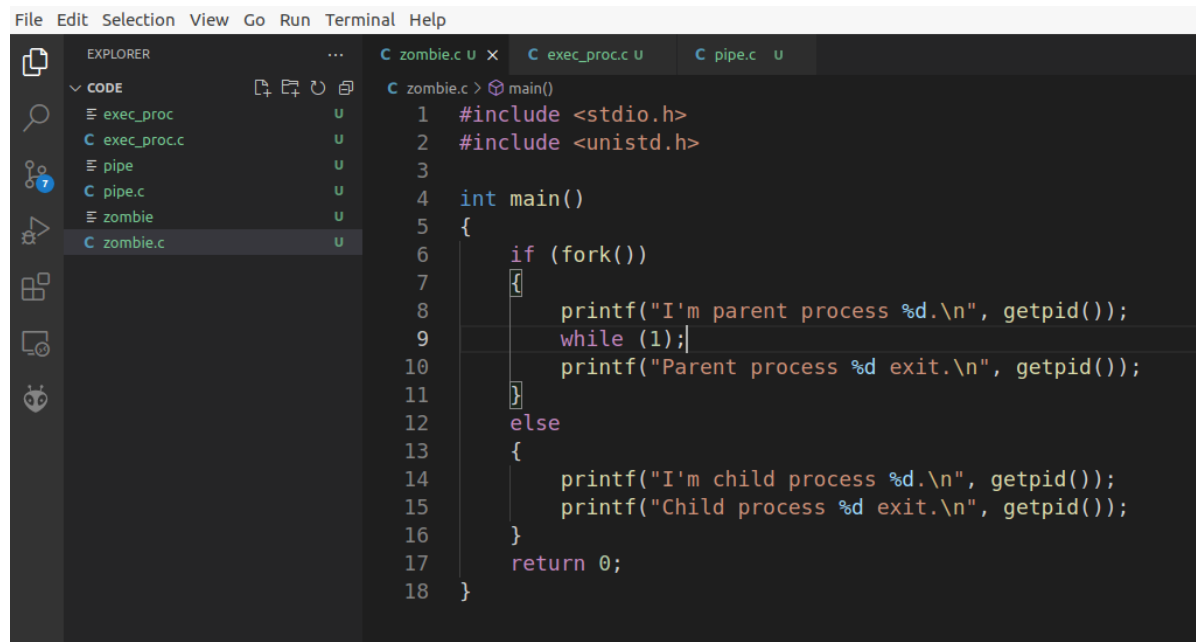
suspends execution of the calling thread until a child specified by pid argument has changed state.

### 3.请回答第四步僵尸进程中列举的第4种情况的结果会是什么？

当父进程结束时，子进程会被过继给init进程或者注册过的祖父进程，在子进程结束后，新的父进程会回收僵尸进程。

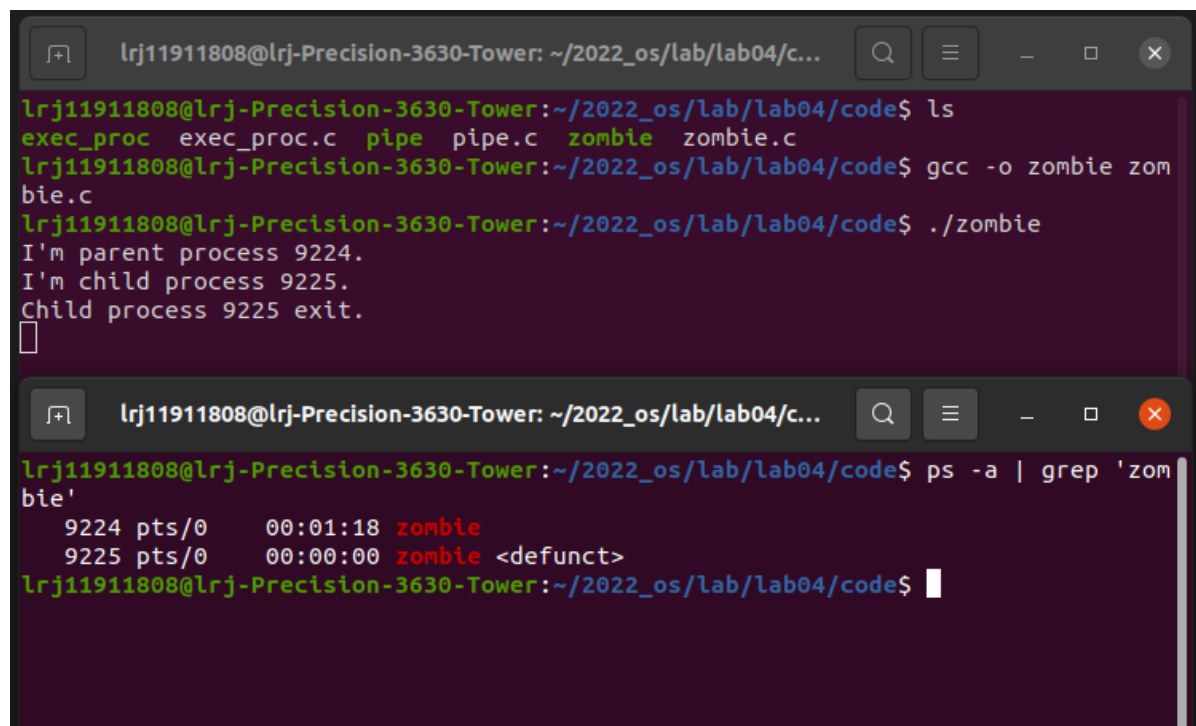
### 4.请编写一段c语言代码（截图），用于产生僵尸进程，并截图僵尸进程的状态(ps)。

代码：



```
File Edit Selection View Go Run Terminal Help
C zombie.c U X C exec_proc.c U C pipe.c U
CODE
exec_proc U
exec_proc.c U
pipe U
pipe.c U
zombie U
zombie.c U
C zombie.c > main()
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     if (fork())
7     {
8         printf("I'm parent process %d.\n", getpid());
9         while (1);
10        printf("Parent process %d exit.\n", getpid());
11    }
12    else
13    {
14        printf("I'm child process %d.\n", getpid());
15        printf("Child process %d exit.\n", getpid());
16    }
17    return 0;
18 }
```

状态：

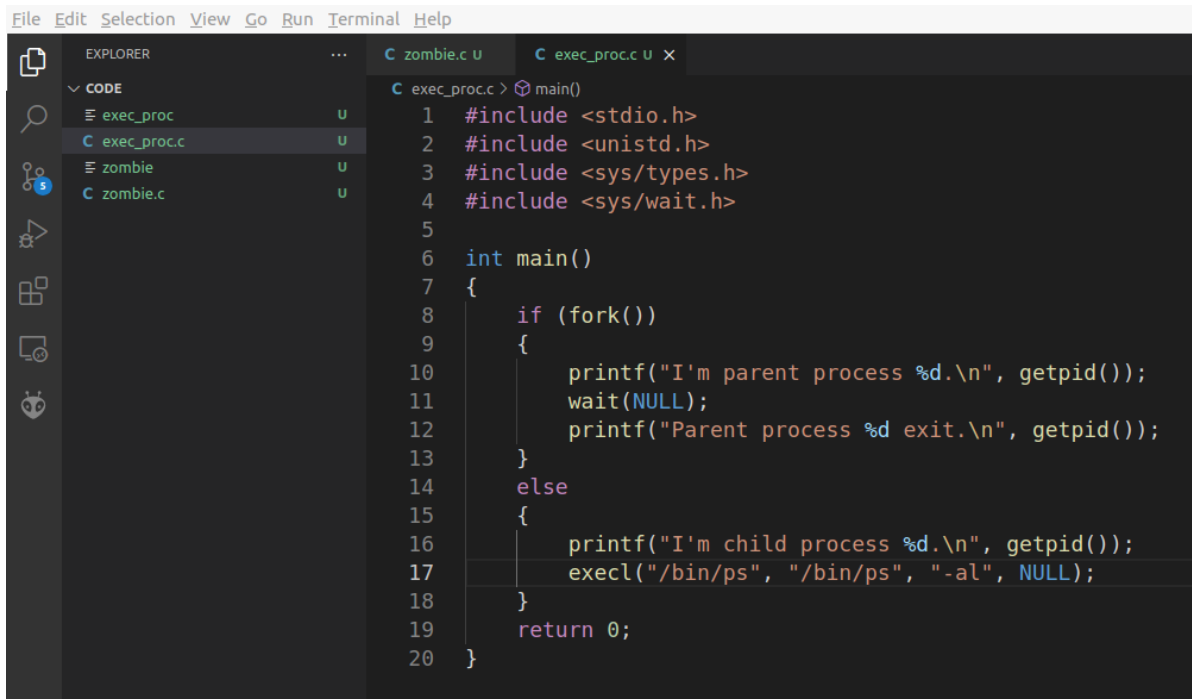


```
lrj11911808@lrj-Precision-3630-Tower: ~/2022_os/lab/lab04/c...
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ ls
exec_proc exec_proc.c pipe pipe.c zombie zombie.c
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ gcc -o zombie zom
bie.c
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ ./zombie
I'm parent process 9224.
I'm child process 9225.
Child process 9225 exit.
█

lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/c...
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ ps -a | grep 'zom
bie'
9224 pts/0    00:01:18  zombie
9225 pts/0    00:00:00  zombie <defunct>
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ █
```

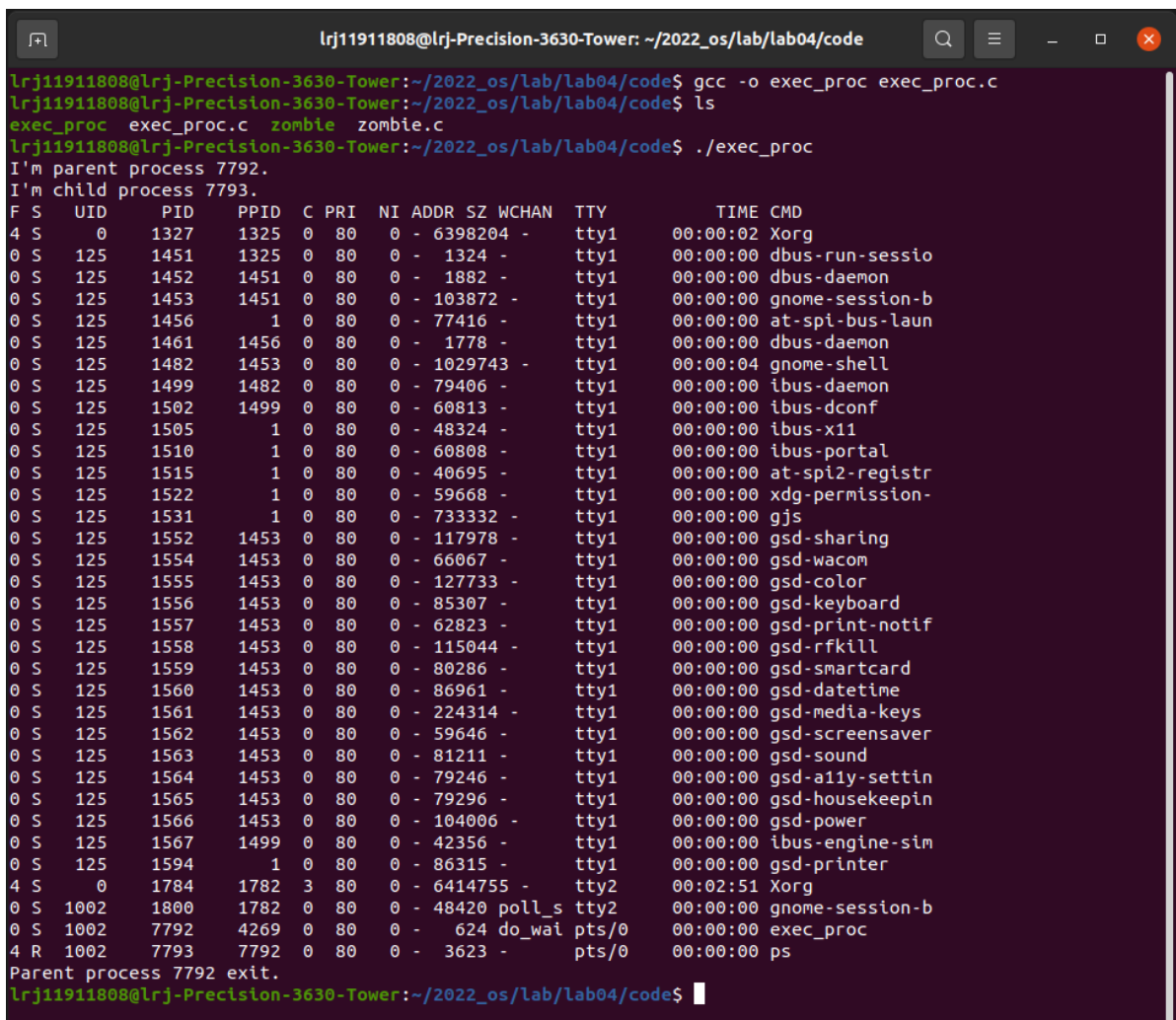
5.请编写一段c语言代码（截图），使程序产生一个子进程，子进程通过exec()实现"ps -al"指令的功能后退出，父进程等待子进程结束后退出，运行效果截图。

代码：



```
File Edit Selection View Go Run Terminal Help
EXPLORER
CODE
  exec_proc
  exec_proc.c
  zombie
  zombie.c
C exec_proc.c U
C exec_proc.c > main()
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5
6  int main()
7  {
8      if (fork())
9      {
10         printf("I'm parent process %d.\n", getpid());
11         wait(NULL);
12         printf("Parent process %d exit.\n", getpid());
13     }
14     else
15     {
16         printf("I'm child process %d.\n", getpid());
17         execl("/bin/ps", "/bin/ps", "-al", NULL);
18     }
19     return 0;
20 }
```

效果：



```
lrj11911808@lrj-Precision-3630-Tower: ~/2022_os/lab/lab04/code
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ gcc -o exec_proc exec_proc.c
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ ls
exec_proc exec_proc.c zombie zombie.c
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$ ./exec_proc
I'm parent process 7792.
I'm child process 7793.
F S  UID      PID     PPID    C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S   0        1327     1325    0  80   0 - 6398204 -   tty1        00:00:02 Xorg
0 S  125       1451     1325    0  80   0 - 1324 -   tty1        00:00:00 dbus-run-sessio
0 S  125       1452     1451    0  80   0 - 1882 -   tty1        00:00:00 dbus-daemon
0 S  125       1453     1451    0  80   0 - 103872 -   tty1        00:00:00 gnome-session-b
0 S  125       1456      1  0  80   0 - 77416 -   tty1        00:00:00 at-spi-bus-laun
0 S  125       1461     1456    0  80   0 - 1778 -   tty1        00:00:00 dbus-daemon
0 S  125       1482     1453    0  80   0 - 1029743 -   tty1        00:00:04 gnome-shell
0 S  125       1499     1482    0  80   0 - 79406 -   tty1        00:00:00 ibus-daemon
0 S  125       1502     1499    0  80   0 - 60813 -   tty1        00:00:00 ibus-dconf
0 S  125       1505      1  0  80   0 - 48324 -   tty1        00:00:00 ibus-x11
0 S  125       1510      1  0  80   0 - 60808 -   tty1        00:00:00 ibus-portal
0 S  125       1515      1  0  80   0 - 40695 -   tty1        00:00:00 at-spi2-registr
0 S  125       1522      1  0  80   0 - 59668 -   tty1        00:00:00 xdg-permission-
0 S  125       1531      1  0  80   0 - 733332 -   tty1        00:00:00 gjs
0 S  125       1552     1453    0  80   0 - 117978 -   tty1        00:00:00 gsd-sharing
0 S  125       1554     1453    0  80   0 - 66067 -   tty1        00:00:00 gsd-wacom
0 S  125       1555     1453    0  80   0 - 127733 -   tty1        00:00:00 gsd-color
0 S  125       1556     1453    0  80   0 - 85307 -   tty1        00:00:00 gsd-keyboard
0 S  125       1557     1453    0  80   0 - 62823 -   tty1        00:00:00 gsd-print-notif
0 S  125       1558     1453    0  80   0 - 115044 -   tty1        00:00:00 gsd-rfkill
0 S  125       1559     1453    0  80   0 - 80286 -   tty1        00:00:00 gsd-smartcard
0 S  125       1560     1453    0  80   0 - 86961 -   tty1        00:00:00 gsd-datetime
0 S  125       1561     1453    0  80   0 - 224314 -   tty1        00:00:00 gsd-media-keys
0 S  125       1562     1453    0  80   0 - 59646 -   tty1        00:00:00 gsd-screensaver
0 S  125       1563     1453    0  80   0 - 81211 -   tty1        00:00:00 gsd-sound
0 S  125       1564     1453    0  80   0 - 79246 -   tty1        00:00:00 gsd-a11y-settin
0 S  125       1565     1453    0  80   0 - 79296 -   tty1        00:00:00 gsd-housekeepin
0 S  125       1566     1453    0  80   0 - 104006 -   tty1        00:00:00 gsd-power
0 S  125       1567     1499    0  80   0 - 42356 -   tty1        00:00:00 ibus-engine-sim
0 S  125       1594      1  0  80   0 - 86315 -   tty1        00:00:00 gsd-printer
4 S   0        1784     1782    3  80   0 - 6414755 -   tty2        00:02:51 Xorg
0 S  1002      1800     1782    0  80   0 - 48420 poll_s tty2        00:00:00 gnome-session-b
0 S  1002      7792     4269    0  80   0 - 624 do_wai pts/0        00:00:00 exec_proc
4 R  1002      7793     7792    0  80   0 - 3623 -   pts/0        00:00:00 ps
Parent process 7792 exit.
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab04/code$
```

## 6.请详细描述手册第七步pipe相关代码中父子进程的功能。

子进程：

- 将本进程 `SIGALRM` 注册为 `write_data` 函数。
- 将本进程 `SIGINT` 注册为 `finish_write` 函数。

父进程：

- 将本进程 `SIGALRM` 注册为 `read_data` 函数。
- 将本进程 `SIGINT` 注册为 `finish_read` 函数。

然后，子进程调用 `kill(getpid(), SIGALRM);`，实际上是调用了 `write_data` 函数，向管道中写入得到的字符串，然后通过 `kill(getppid(), SIGALRM);` 调用父进程的 `read_data` 函数，父进程在 `read_data` 函数中从管道读取子进程写入的字符串，然后通过 `kill(pid, SIGALRM);` 调用子进程的 `write_data` 函数，如此循环，读写往复。

如果我们从terminal按下 `ctrl+c`，则会通过 `SIGINT` 触发父进程和子进程的handler，即 `finish_write` 和 `finish_read` 函数，关闭管道的读写并结束进程。