

同济大学计算机系

操作系统课程实验报告



学 号 2251557

姓 名 代文波

专 业 计算机科学与技术

授课老师 方钰

实验五：UNIX V6++中新进程创建与父子进程同步

一、实验目的

结合课程所学知识，通过在 UNIX V6++ 实验环境中编写使用了父进程创建子进程的系统调用 fork，进程终止及父子进程同步的系统调用 exit 和 wait 的应用程序，并观察他们的运行结果，进一步熟悉 UNIX V6++ 中关于进程创建、调度、终止和撤销的全过程，实践 UNIX 中最基本的多进程编程技巧。

二、实验设备及工具

已配置好的 UNIX V6++ 运行和调试环境。

三、预备知识

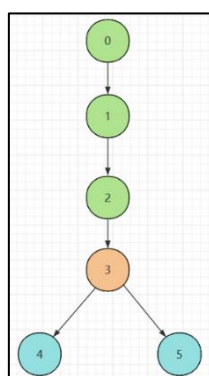
- (1) 熟悉如何在 UNIX V6++ 中编译、调试和运行一个用户编写的应用程序。
- (2) 熟练掌握 UNIX V6++ 进程管理的相关算法与实施细节。
- (3) 熟悉 fork, exit, wait 和 sleep 四个系统调用的执行过程。

四、实验内容

第一部分：完成实验 4.1 ~ 4.3，建立符合要求的进程树，并通过父进程是否执行 wait，执行几个 wait 来调整父子进程之前的同步顺序，实现父进程等待所有子进程、父进程先于所有子进程和父进程先于部分子进程等场景，截图展示程序运行结果；

4.1 添加一个名为 procTest.exe 的可执行程序。

要求 procTest.exe 程序通过 fork 系统调用，创建出如下图所示的进程树。



4.1.1 在 program 文件中加入一个名为 procTest.c 的文件

```
programs > C procTest.c > main10
1  #include <stdio.h>
2  #include <sys.h>
3  int main1()
4  {
5      int ws = 2;
6      int i,j,k,pid,ppid;
7      if(fork())//2#创建3#, 2#执行if, 3#执行else
8      {
9          //2
10         sleep(2);
11         for(k=1;k<6;k++)
12         {
13             printf("%d,%d;",k,getppid(k));
14         }
15         printf("\n");
16     }
17     else//2#创建的3#执行
18     {
19         //3
20         if(fork())//3#创建4#, 3#执行if, 4#执行else
21         {
22             if(fork())//3#创建5#, 3#执行if, 5#执行else
23             {
24                 //3#
25                 pid=getpid();
26                 ppid=getppid(pid);
27                 for(k=0;k<ws;k++)
28                 {
29                     i=wait(&j);
30                     printf("Process %d#:My child %d is finished with exit status %d\n",pid,i,j);
```

```
programs > C procTest.c > main10
3  int main1()
31
32     }
33     printf("Process %d# finished: My father is %d\n",pid,ppid);
34     exit(ppid);
35 }
36 else//3#创建的5#执行else
37 {
38     //5
39     pid=getpid();
40     ppid=getppid(pid);
41     printf("Process %d# finished: My father is %d\n",pid,ppid);
42     exit(ppid);
43 }
44 else//3#创建的4#执行else
45 {
46     //4#
47     pid=getpid();
48     ppid=getppid(pid);
49     printf("Process %d# finished: My father is %d\n",pid,ppid);
50     exit(ppid);
51 }
52 }
53 }
```

4.1.2 重新编译运行 UNIX V6++ 代码

```
问题 输出 调试控制台 终端 端口
● [vesper_center_279@archlinux unix-v6pp-tongji]$ make all
```

```
问题  输出  调试控制台  终端  端口

[bin/..] > [info 9] 创建文件夹: etc
[bin/..] > [info] 切换路径。
[bin/./etc] > [info 5] 上传成功: v6pp_splash.bmp
[bin/./etc] > [info] 切换路径。
[bin/./etc/..] > [info 5] 上传成功: Shell.exe
[bin/./etc/..] > bye!
cp target/img-workspace/c.img target/
build success (unix-v6pp-tongji).
[vesper_center_279@archlinux unix-v6pp-tongji]$
```

4.1.3 程序运行结果

```
QEMU - Press Ctrl+Alt+G to release grab

Machine  View

welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 5# finished: My father is 3
Process 3#:My child 4 is finished with exit status 3
Process 3#:My child 5 is finished with exit status 3
Process 3# finished: My father is 2
1,0;2,1;3,2;4,-1;5,-1;
[/bin]#
```

【注意】这里一旦先调用 ls 展示文件夹目录，则 ls 自己会作为一个进程，编号为 2，导致后面的 procTest 的进程号会变成 3 号，进而导致程序预测结果都向后移动一位，进而造成有差异的程序运行结果如下：

```
QEMU - Press Ctrl+Alt+G to release grab

Machine  View

welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#ls
Directory '/bin':
test  fork  mkdir  stack  showStack  rr      sigTest  performance  trace  cp  proc
Test  date  forks  ralloc  cat      sig     shutdown  echo  testSTDOUT  getppid copy
file  newsig  ls
[/bin]#procTest
Process 5# finished: My father is 4
Process 6# finished: My father is 4
Process 4#:My child 5 is finished with exit status 4
Process 4#:My child 6 is finished with exit status 4
Process 4# finished: My father is 3
1,0;2,-1;3,1;4,3;5,-1;
[/bin]#
```

4.2 父进程先于所有子进程结束

如果我们不希望 3#进程等待 4#进程和 5#进程结束后再结束，而是希望 3#进程先于两个子进程结束，只要 3#进程不执行 wait 操作即可，即需将代码中 ws 的值改为 0。

4.2.1 代码修改

```
programs > C procTest.c > main10
1  #include <stdio.h>
2  #include <sys.h>
3  int main1()
4  {
5      int ws = 0;
6      int i,j,k,pid,ppid;
7      if(fork())//2#创建3#, 2#执行if, 3#执行else
8      {
9          //2
10         sleep(2);
11         //这里的“6”是因为一共有六个进程0-5
12         for(k=1;k<6;k++)
13         {
14             printf("%d,%d;",k,getppid(k));
15             //若进程已终止, 则其进程号不存在, getppid返回-1
16         }
17         printf("\n");
18     }
```

4.2.2 重新编译运行 UNIX V6++ 代码

```
问题 输出 调试控制台 终端 端口 + v
• [vesper_center_279@archlinux unix-v6pp-tongji]$ make all

问题 输出 调试控制台 终端 端口 + v

[bin] > [info] 切换路径。
[bin/..] > [info 9] 创建文件夹: etc
[bin/..] > [info] 切换路径。
[bin/../etc] > [info 5] 上传成功: v6pp_splash.bmp
[bin/../etc] > [info] 切换路径。
[bin/../etc/..] > [info 5] 上传成功: Shell.exe
[bin/../etc/..] > bye!
cp target/img-workspace/c.img target/
build success (unix-v6pp-tongji).
```

4.2.3 程序运行结果

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View

welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 3# finished: My father is 2
Process 4# finished: My father is 1
Process 5# finished: My father is 1
1,0,2,1;3,2,4,-1;5,-1;
[/bin]#
```

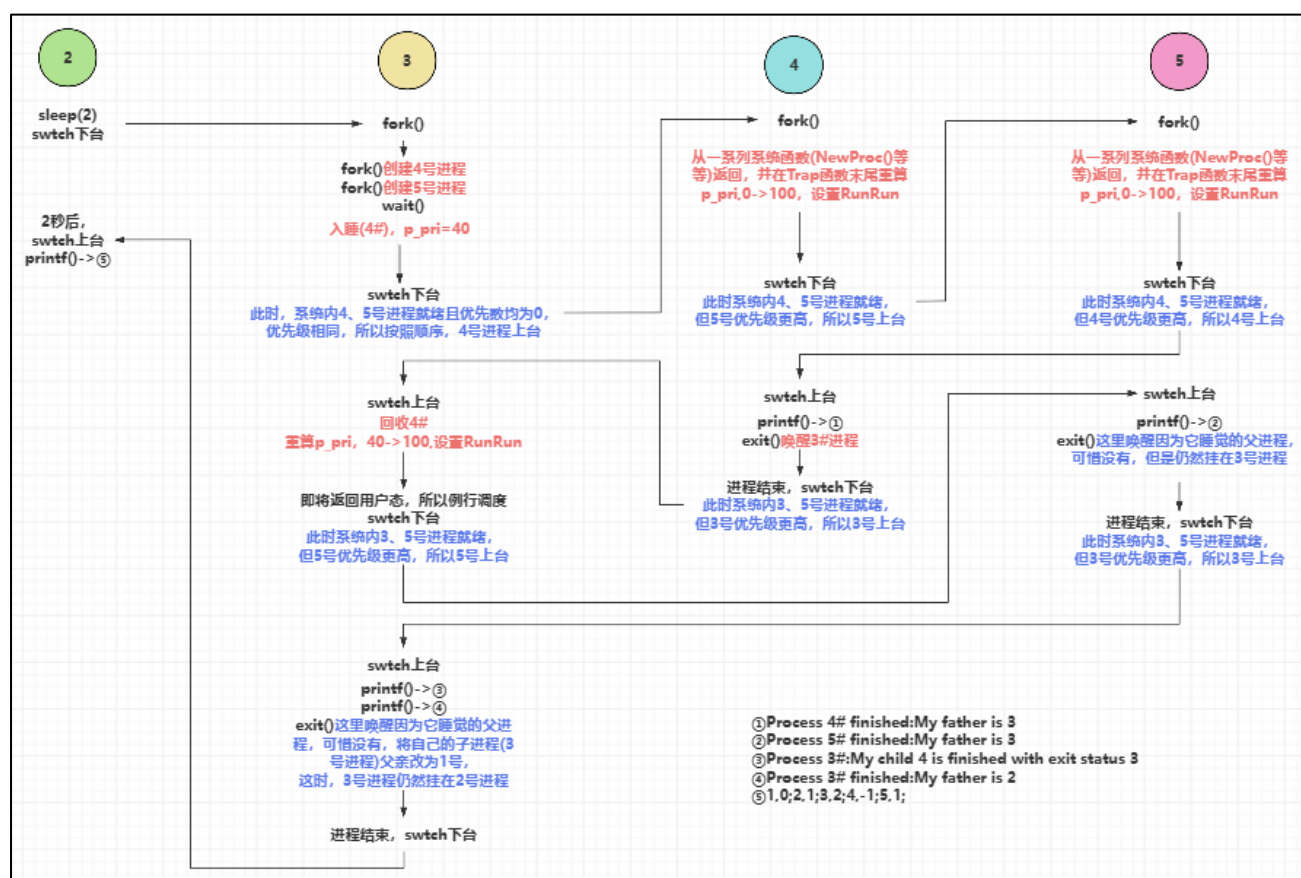
第二部分：

4.3. 父进程先于部分子进程结束

通过前述两份代码，我们可以发现，执行几次 `wait`，父进程就可以接收并处理几个终止的子进程。这里我们可以尝试只删除代码 1 中的一个 `wait`，而不是两个都删除。这样父进程将只等待其中的一个子进程，于是我们可以得到如下图所示的输出。

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 5# finished: My father is 3
Process 3#:My child 4 is finished with exit status 3
Process 3# finished: My father is 2
1,0;2,1;3,2;4,-1;5,1;
[/bin]#
```

4.3.1 绘图解释



4.3.2 问题解答

(1) 为什么 4#进程和 5#进程终止时，都能找到自己的父进程是 3#进程？

答：①4#进程结束时，3#进程因 wait()一直在睡觉，所以 4#进程终止时可以找到父进程是 3#；
②对于 5#进程，3#此前因 4#唤醒而回收 4#，并重算了 3#自己的优先数并设置 RunRun>0。在 3#进程在 wait()返回用户态时的例行调度中，5#进程抢占上台，所以尽管 3#进程没有用 wait()等待它，但是此时 3#进程仍然存在，尚未消失，所以 5#进程终止时可以找到父进程是 3#进程。

(2) 最后的打印输出中，5#进程的父进程是 1#，说明 5#进程此时还是存在的，为什么？5#进程的图象将由谁在什么时间回收？

答：①5#进程结束时调用了 exit()，想唤醒因为它睡觉的父进程来回收自己，可惜其父进程 3#进程没有因为它入睡，所以并未成功唤醒（顺便提一下，3#进程此时处于就绪状态），进而 5#进程只能清空内存并将 user 结构搬到盘交换区后终止进程，但是 5#进程仍然挂在 3#进程上。后来，3#进程结束时，将自己的子进程的父进程都设置为 1#进程，这时 5#进程的父进程就变成了 1#进程，所以在 2#进程最后的打印输出中，5#进程还存在并且其父进程是 1#。②5#进程的图像最后会由 1 号进程在 2#进程执行完毕后通过 swtch 上台后回收。

第三部分：

4.4. 抢占父进程

4.4.1 为什么 4.1~4.3 的实验中，父进程 3#进程始终没有被抢占？在本实验的代码中，如果父进程 3#进程不执行 wait，可以被子进程抢占的时机和条件是什么？

答：第一问：对于 4.1~4.3 的实验，3#进程一方面执行时间太短，没有执行时钟中断的复杂部分，一方面没有因中断唤醒优先级更高的进程，最后一方面执行过程中没有因调用输入输出设备等原因睡眠而下台。

第二问：如果如果父进程 3#进程不执行 wait，可以被子进程抢占的时机和条件如下：

① 时机：一秒结束，3#重算自己的优先数，必然会设置 RunRun，进而例行调度中可以抢占；

条件：3#进程上台有足够多的事情去做，待够较长时间并且时钟中断来临时 `Time::lbolt` 满足让该进程执行复杂任务的条件（课上讲的是 `Time::lbolt>60`,UNIX V6++ 中代码写的是 `Time::lbolt>120`）；

② 时机：给 3#进程安排中断，让 3#唤醒拥有更高优先级的进程而设置 RunRun 后，最后中断返回时的例行调度。

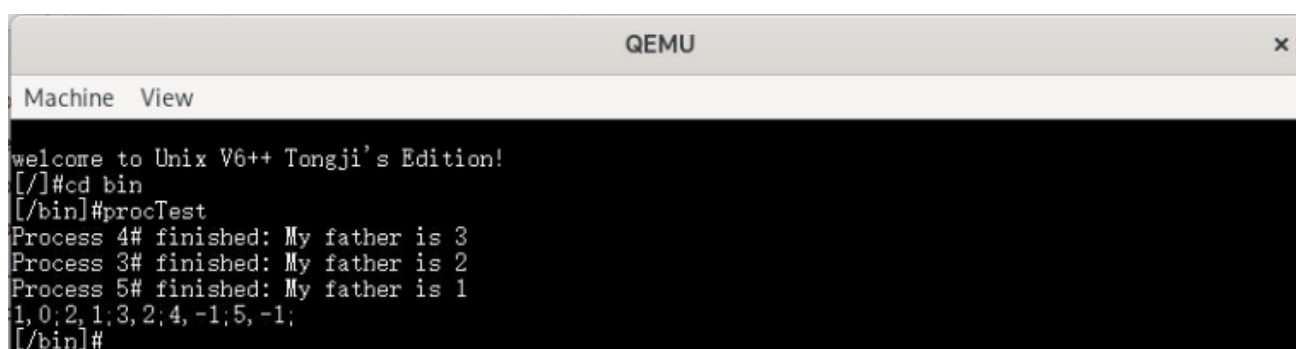
条件：3#进程响应中断时，必须唤醒比自己上台时优先级更高的进程进而设置 RunRun

③ 时机：让 3#进程因调用输入输出设备等原因睡眠进而下台。

条件：3#进程需要因为一些原因睡眠而主动下台让出 cpu。

4.4.2 先将 3#进程执行代码部分的两次 wait 操作删除，再尝试两种修改代码的方案，创造 3#进程可以被抢占的机会，进而得到如图 6 和图 7 所示的执行结果。

1、情况一



```
QEMU x
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 3# finished: My father is 2
Process 5# finished: My father is 1
1, 0; 2, 1; 3, 2; 4, -1; 5, -1;
[/bin]#
```

(1) 修改代码：

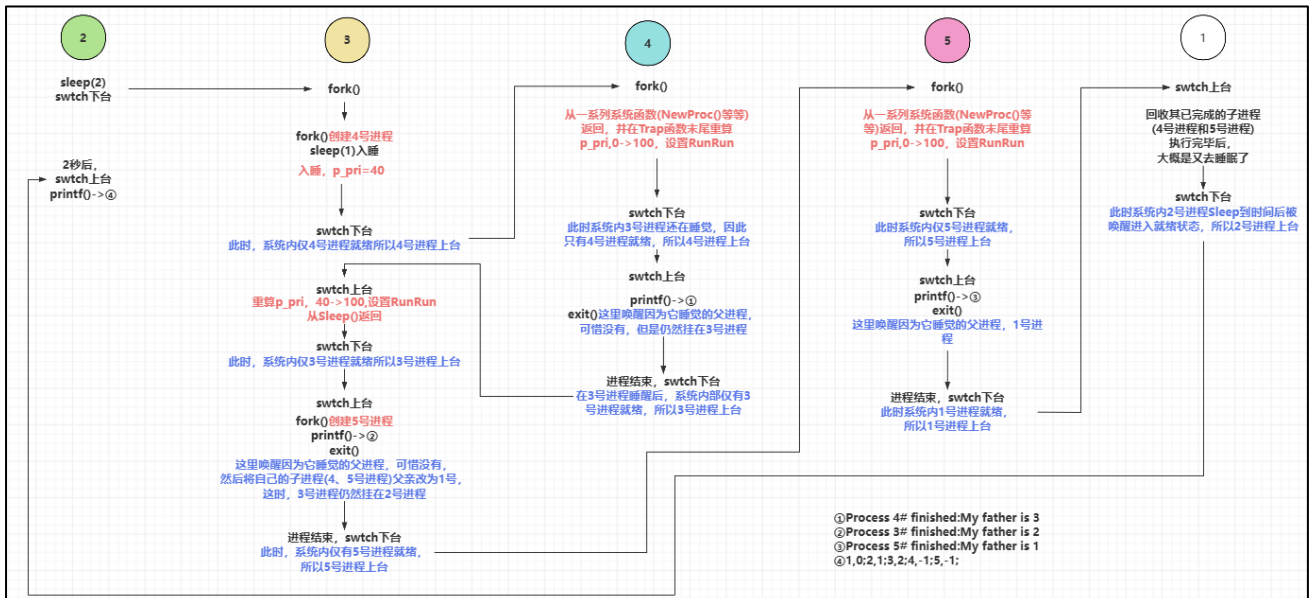
在 3#创建 4#、5#进程之间调用 Sleep 进而让其主动下台。


```

programs > C procTest.c > main1()
1  #include <stdio.h>
2  #include <sys.h>
3  int main1()
4  {
5      int ws = 0;
6      int i,j,k,pid,ppid;
7      if(fork())//2#创建3#, 2#执行if, 3#执行else
8      {
9          //2#
10         sleep(2);
11         //这里的“6”是因为一共有六个进程0-5
12         for(k=1;k<6;k++)
13         {
14             printf("%d,%d;",k,getppid(k));
15             //若进程已终止, 则其进程号不存在, getppid返回-1
16         }
17         printf("\n");
18     }
19     else//2#创建的3#执行
20     {
21         //3#
22         if(fork())//3#创建4#, 3#执行if, 4#执行else
23         {
24             sleep(1);//第一问改动的地方!
25             if(fork())//3#创建5#, 3#执行if, 5#执行else
26             {
27                 //3#
28                 pid=getpid();
29                 ppid=getppid(pid);
30                 for(k=0;k<ws;k++)
31                 {
32                     i=wait(&j);
33                     //这里需要注意: wait是系统的睡眠函数, 返回时相当于核心态返回用户态, 在RunRun>0时会进行例行调度

```

(2) 绘图解释



情况二：

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View

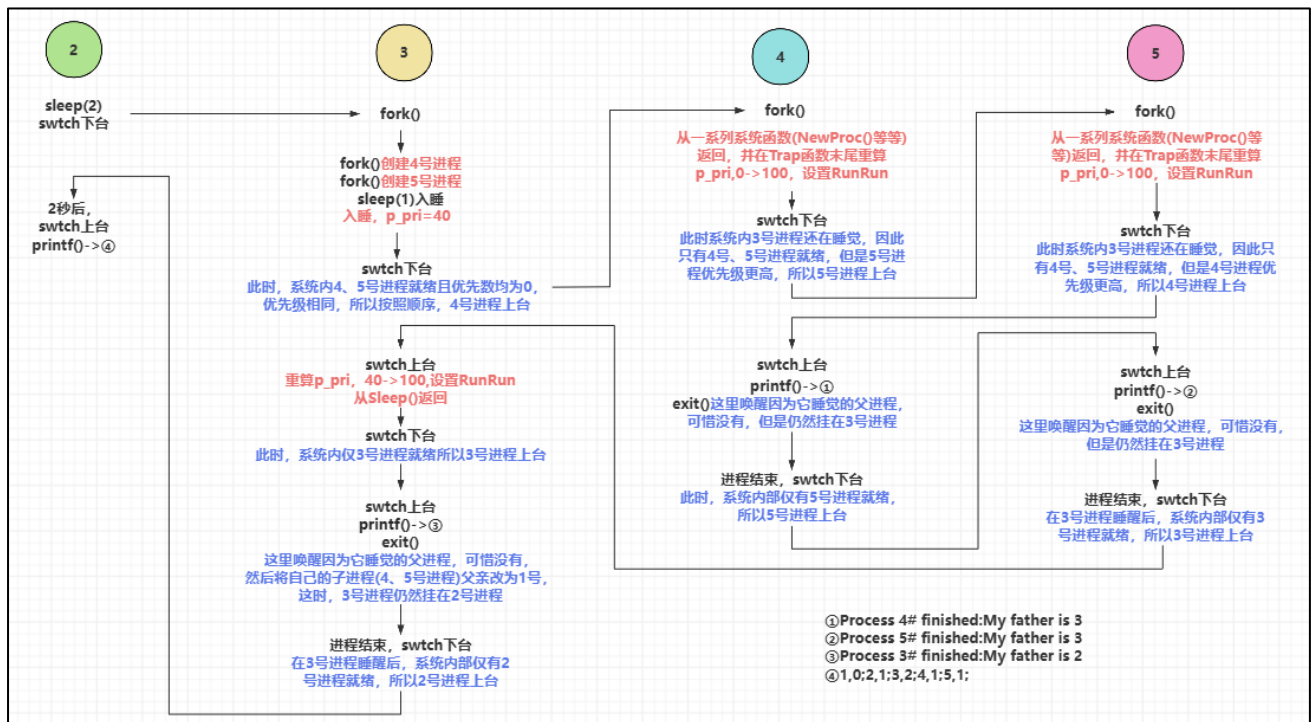
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 5# finished: My father is 3
Process 3# finished: My father is 2
1,0;2,1;3,2;4,1;5,1;
[/bin]#
```

(1) 修改代码

在 3 号进程创建完 4、5 号进程后调用 Sleep 进而让其主动下台。

```
programs > C procTest.c > main1()
1  #include <stdio.h>
2  #include <sys.h>
3  int main1()
4  {
5      int ws = 0;
6      int i,j,k,pid,ppid;
7      if(fork())//2#创建3#, 2#执行if, 3#执行else
8      {
9          //2#
10         sleep(2);
11         //这里的“6”是因为一共有六个进程0-5
12         for(k=1;k<6;k++)
13         {
14             printf("%d,%d;",k,getppid(k));
15             //若进程已终止，则其进程号不存在，getppid返回-1
16         }
17         printf("\n");
18     }
19     else//2#创建的3#执行
20     {
21         //3#
22         if(fork())//3#创建4#, 3#执行if, 4#执行else
23         {
24             // sleep(1);//第一问改动的地方!
25             if(fork())//3#创建5#, 3#执行if, 5#执行else
26             {
27                 sleep(1);//第二问改动的地方!
28                 //3#
29                 pid=getpid();
30                 ppid=getppid(pid);
31                 for(k=0;k<ws;k++)
32                 {
33                     i=wait(&j);
34                     //这里需要注意: wait是系统的睡眠函数，返回时相当于核心态返回用户态，在RunRun>0时会进行例行调度
35                     printf("Process %d#My child %d is finished with exit status %d\n",pid,i,j);
```

(2) 绘图解释



(3) 问题解答

观察图 6（情况一）和图 7（情况二）不难发现，图 6 中 3# 仅被 4# 进程抢占，而图 7 中，4# 和 5# 均抢占了 3#。因为被抢占的位置不同，导致程序最后输出时，图 6 中 4# 和 5# 已被撤销，而图 7 中，4# 和 5# 显示父进程为 1#，为什么？

答：（1）图 6（即情况 1 中），3# 创建好 4# 后就入睡了，主动让出 cpu。此时，系统内仅 4# 就绪，所以 4# 进程上台执行。4# 在 `fork()` 的 Trap 末尾重算优先数，设置 RunRun，在 `fork()` 返回用户态时例行调度，但此时系统内仍旧只有 4# 就绪，所以 4# 进程再次上台。4# 再次上台后，打印输出，调用 `exit()` 删除内存、保存 user 到盘交换区。因没有唤醒因为它睡的父亲进程回收自己，所以结束时仍挂在 3# 上。之后 3# 睡醒上台创建好 5#，打印输出，调用 `exit()` 删除内存、保存 user 到盘交换区、将子进程 4#、5# 交给 1#。因为没有唤醒因为它睡的父亲进程回收自己，所以结束时仍挂在 2# 上。之后 5# `swtch` 上台，在 `fork()` 的 Trap 末尾重算优先数，设置 RunRun，在 `fork()` 返回用户态时例行调度，但此时系统内仍旧只有 5# 就绪，所以 5# 进程再次上台。5# 再次上台后，打印输出，调用 `exit()` 删除内存、保存 user 到盘交换区。但是唤醒了因为它睡的父亲进程回收自己，所以结束时仍挂在 3# 上。

觉的 1#，之后 1#上台将其已经结束的进程 4#、5#回收。综上，程序最后输出时，图 6 中 4# 和 5#已被撤销。

(2) 图 7 (情况 2) 中，3#在创建完 4#、5#进程后入睡。4#、5#进程先依次上台进行 fork() 返回，然后再依次上台打印输出，调用 exit()删除内存、保存 user 到盘交换区，因为没唤醒因自己睡觉的父进程而结束时仍挂在 3#上。之后，3#睡醒后上台，打印输出，调用 exit 删除内存、保存 user 到盘交换区、将其子进程(4#、5#)的父进程都设置成为了 1#，因为没唤醒因自己睡觉的父进程而结束时仍挂在 2#上。最后，2#上台打印信息，这时因为 1#尚未上台，所以 4#、5#作为其已经完成的子进程还没有被收回，进而导致图 7 中，4#和 5#显示父进程为 1#。