

# 进程的创建和终止

同济大学计算机系 操作系统作业  
学号:2151769

姓名: 吕博文

2023-12-7

## 一、例 1

### 四、wait、exit系统调用的使用方法，例1

```
#include <stdio.h>
#include <sys.h>
int main1(int argc, char* argv[])
{
    int i, j;
    if(fork())
    {
        i = wait(&j);
        printf("It is parent process. \n");
        printf("The finished child process is %d. \n", i);
        printf("The exit status is %d. \n", j);
    }
    else
    {
        printf("It is child process. \n");
        sleep( 2 );
    }
}
```

已知，Unix V6++系统，只有这一个程序在执行。父进程pid==2，子进程pid==3。

问，

- (1) 父进程会睡吗？
- (2) 这个程序的输出是什么？
- (3) 删除sleep(2)，父进程有可能不睡吗？
- (4) 无论有没有sleep(2)，这个程序的输出是确定的，断言正确否？为什么？

操作系统

电信学院计算机系 邓鑫

39

- (1) 父进程执行 wait() 系统调用一定会睡。
- (2) 这个程序的输出是：  
It is child process.  
It is parent process.  
The finished child process is 3  
The exit status is 0.
- (3) 删除 sleep(2)，父进程有可能不睡。fork 之后子进程先执行，父进程就不会睡。  
细节如下：如果 fork 创建子进程之后，子进程先运行。输出 It is child process 之后，子进程终止[注]。待父进程执行 wait 系统调用时，存在已终止的子进程，无需入睡等待，直接回收子进程 PCB。  
[注]子进程终止时，系统会唤醒父进程。这是个无效的唤醒操作，对系统不会产生任何影响。
- (4) 正确。因为子进程不终止，父进程通不过 wait 系统调用。

\*\*\*\*\* 这个程序的运行过程如下：

父进程 fork 创建子进程，成功后，有可能父进程先运行，也有可能子进程先运行。

若父进程先运行，它会执行 wait 系统调用入睡。

入睡后，系统选中子进程。

子进程输出，执行 sleep 系统调用，放弃 CPU 后，系统没有就绪进程了，进

入 idle 状态。0#进程等中断。

2s 后，子进程被时钟中断唤醒，sleep 系统调用返回。

子进程终止，唤醒父进程。

父进程被唤醒后，wait 系统调用返回，回收子进程 PCB，获得终止子进程的 pid(3) 和终止码 (0)，赋给 i 变量和 j 变量。

父进程输出，之后终止，唤醒它的父进程(shell 进程)。shell 进程回收父进程 PCB，之后输出命令行提示符 #，等待用户输入下个命令行。\*\*\*\*\*程序执行，到此结束。

若子进程先运行，它会输出，执行 sleep 系统调用入睡。

入睡后，选中父进程。

父进程执行 wait 系统调用，入睡。放弃 CPU 后，系统没有就绪进程了，进入 idle 状态。0#进程等中断。

2s 后，子进程被时钟中断唤醒，sleep 系统调用返回。

子进程终止，唤醒父进程。

父进程被唤醒后，wait 系统调用返回，回收子进程 PCB，获得终止子进程的 pid(3) 和终止码 (0)，赋给 i 变量和 j 变量。

父进程输出，之后终止，唤醒它的父进程(shell 进程)。shell 进程回收父进程 PCB，之后输出命令行提示符 #，等待用户输入下个命令行。\*\*\*\*\*程序执行，到此结束。

2 种情况，程序的执行过程极为类似，区别仅在下划线标出的部分。

## 二、例 2

### 四、wait、exit 系统调用的使用方法

```
#include <stdio.h>
#include <sys.h>
int main1(int argc, char* argv[])
{
    int i, j;
    if(fork())
    {
        sleep(2);
        i = wait(&j);
        printf("It is parent process. \n");
        printf("The finished child process is %d. \n", i);
        printf("The exit status is %d. \n", j);
    }
    else
    {
        printf("It is child process. \n");
    }
}
```


已知，Unix V6++ 系统，只有这一个程序在执行。父进程 pid==2，子进程 pid==3。

问，(1) 父进程会睡吗？  
(2) 这个程序的输出是什么？  
(3) 系统里有几个进程？

操作系统 电信学院计算机系 邓鑫 40

- (1) 父进程一定不睡，因为父进程会睡 2s。执行 wait 系统调用时，子进程已经终止了。
- (2) 这个程序的输出和上一题一样。
- (3) Unix V6++ 系统，有 4 个进程：0#进程，1#进程（就是 shell 进程），父进程（2#进程）和子进程（3#进程）。

### 三、例 3。有改动。



## wait、exit系统调用的使用方法，例3

```
#include <stdio.h>
#include <sys.h>
int main1(int argc, char* argv[])
{
    int i, j;
    if(fork())
    {
        printf("father. \n");
        if(fork())
        { i = wait(&j); printf("exit child: %d exit status: %d. \n", i, j>>8); }
        else
        { printf("second child. \n"); exit(3); }
    }
    else
    {
        sleep( 2);
        printf("first child. ppid: %d \n", getppid( ));
    }
}
```

已知， Unix V6++系统，只有这一个程序在执行。父进程 pid==2，子进程1 pid==3，子进程2 pid==4。

问，（1）这个程序的输出是什么？  
（2）子进程1 和 子进程2 的PCB分别是哪个进程回收的？  
（3）父进程终止后，PCB是哪个进程回收的？

操作系统电信学院计算机系 邓蓉50

(1)

father.  
second child.  
exit child: 4. exit status: 3.  
first child. ppid: 1.

最后一个输出，是因为父进程终止时，将子进程的 ppid 改为 1#进程。

(2)

子进程 1 的 PCB 是 1#进程回收的。子进程 2 的 PCB 是父进程回收的。

(3)

父进程终止后，PCB 是 shell 进程回收的。

[注] Unix V6++是单用户系统。只有一个终端，所以现在只需要一个 shell 进程。1#进程就是 shell 进程。多用户系统不可以这样，有多少个用户同时上机，就有多少个 shell 进程，1#进程是这些 shell 进程的父进程。

四、下面的这个程序会输出几个整数？请写出程序的输出，并请在代码中标出父进程执行的所有语句和子进程执行的所有语句。

```
L1: #include <stdio.h>
L2: void main (void )
L3: {
L4:     int i = 10, x ;
L5:     if ( x = fork() )
L6:     {
L7:         i += 10;
L8:         printf ( "%d\t", i );
```

```

L9:      }
L10:     else
L11:         printf ( "%d\t", i );
L12:     printf ( "%d\t", i );
L13: }

```

参考答案：这个程序会输出 4 个整数。这是因为语句 L12 执行了 2 次。

程序的输出：

可能是：	10	20	10	20
可能是：	10	20	20	10
还可能是：	20	10	20	10
或：	20	10	10	20
或：	20	20	10	10
或：	10	10	20	20

## 习题部分：

### 阅读程序，回答问题

代码 1.1。假定父进程的PID是 007，子进程的PID是008。写出程序的输出。

```

#include <stdio.h>
#include <sys.h>
main()
{
    int i=10,j =20;
    if( i=fork() )
    {
        printf("It is parent process. PID = %d, i = %d\n",getpid( ), i);
        i=wait(&j);
        printf("The finished child process is %d. \n", i);
        printf("The exit status is %d. \n", j);
    }
    else
    {
        printf("It is child process. PID = %d, i = %d\n",getpid( ), i);
        exit(1);
    }
}

```

解：（1）假定父进程执行完 fork 后是父进程先上台执行，那么首先进入 if 语句执行第一句 printf，之后父进程 wait 入睡等待，之后子进程上台执行 else 语句块中的

printf 语句之后执行 exit 退出，之后父进程上台接着执行剩下的两句 printf 语句，最终结果如下：

It is parent process. PID = 007,i = 008

It is child process. PID = 008,i = 0

The finished child process is 008

The exit status is 1 (**exit status 是调入参数左移 8 位后的结果**) 256

(2) 假定父进程执行完 fork 后子进程先上台执行，那么首先进入 else 语句块执行 printf 语句，之后子进程 exit 终止，之后父进程上台执行，首先执行第一句 printf 语句，执行 wait 因为子进程已经终止，所以直接回收 PCB，截止执行下面两句 printf 语句，输出结果如下：

It is child process. PID = 008,i = 0

It is parent process. PID = 007,i = 008

The finished child process is 008

The exit status is 1 (**exit status 是调入参数左移 8 位后的结果**) 256

代码 1.2。假定父进程的PID是 007，子进程的PID是008。

```
#include <stdio.h>
#include <sys.h>
main()
{
    int i=10,j=20;
    if( i=fork() )
    {
        printf("It is parent process. PID = %d, i = %d\n",getpid(), i);
    }
    else
    {
        sleep(100);
        printf("It is child process. PID = %d, i = %d\n",getpid(), i);
        exit(1);
    }
}
```

(1) 写出程序的输出。

It is parent process. PID = 007,i = 008

It is child process. PID = 008,i = 0

(2) T0时刻，父进程创建子进程。子进程何时终止？终止后，子进程的PCB何时回收，

由谁来回收。

T0时刻，父进程创建子进程，之后可能父进程首先上台执行结束之后子进程再上台执行，也可能子进程先上台执行但是执行sleep函数入睡之后也是父进程首先执行完毕，父进程结束时会将未结束的子进程的父进程修改为1号进程，T0+100时刻，子进程被唤醒执行完剩下的语句之后，调用exit结束，由1号进程回收子进程的PCB。

代码 1.3。假定父进程的PID是 007，子进程的PID是008。

```
#include <stdio.h>
#include <sys.h>
main()
{
    int i=10, j=20;
    if( i=fork() )
    {
        printf("It is parent process. PID = %d, i = %d\n", getpid( ), i);
        sleep(100);
    }
    else
    {
        printf("It is child process. PID = %d, i = %d\n", getpid( ), i);
        exit(1);
    }
}
```

(1) 写出程序的输出。

父进程先上台执行：

It is parent process. PID = 007,i = 008

It is child process. PID = 008,i = 0

子进程先上台执行：

It is child process. PID = 008,i = 0

It is parent process. PID = 007,i = 008

(2) T0时刻，父进程创建子进程。printf耗时忽略。子进程的PCB何时回收，由谁来回收。

T0时刻父进程创建子进程，若父进程首先上台执行，那么执行完printf语句之后执行sleep入睡，之后子进程上台执行，执行完语句之后调用exit结束，等待父进程唤醒之后在T0+100时刻回收其PCB；若子进程首先上台执行，那么执行完语句调用exit退出，父进程上台执行，执行完printf语句之后sleep入睡，T0+100时刻唤醒之后父进程回收子进程PCB

子进程终止的时候，exit 叫不醒父进程\*。所以，无论哪种情况，子进程的 PCB 一定要等 100s 以后，父进程终止时，由 1#进程回收。

注：

exit。wakeupall（父进程 008 的 process 对象的起始地址）

父进程 008 sleep 入睡，睡在 闹钟变量 tout 上

代码 1.4。假定父进程的PID是 007，第一个子进程的PID是008，第二个子进程的PID是 009。

```
#include <stdio.h>
#include <sys.h>
main()
{
    int i=10,j=20;
    if( i=fork() )
    {
        printf("It is parent process. PID = %d, First Son: %d\n", getpid( ), i);
        if( i=fork() ) {
            printf("It is parent process. PID = %d, Second Son: %d\n", getpid( ), i);
            i = wait( &j );
            printf("Exit Son: %d. Exit Status= %d\n", i, j);
        }
        else {
            printf("It is child process. PID = %d, i = %d\n", getpid( ), i);
            exit( 2 );
        }
    }
}
```

```

        }
    }
    else
    {
        sleep(100);
        printf("It is child process. PID = %d, i = %d\n", getpid( ), i);
        exit(1);
    }
}

```

(1) 写出程序的输出。

若父进程创建第二个子进程之后父进程先执行：

It is parent process. PID = 007, First Son = 008

It is parent process. PID = 007, Second Son = 009

It is child process. PID = 009, i = 0

Exit Son : 009, Exit Status : 2

It is child process. PID = 008, i = 0

若父进程创建第二个子进程之后第二个子进程先执行：

It is parent process. PID = 007, First Son = 008

It is child process. PID = 009, i = 0

It is parent process. PID = 007, Second Son = 009

Exit Son : 009, Exit Status : 2

It is child process. PID = 008, i = 0

(2) T<sub>0</sub>时刻，父进程创建子进程。printf耗时忽略。子进程的PCB何时回收，由谁来回收。

T<sub>0</sub>时刻父进程创建第一个子进程，假设此时第一个子进程首先上台执行，那么其执行sleep入睡，父进程继续上台执行完第一个printf语句之后，继续创建第二个



子进程，如果此时父进程继续上台执行，那么执行完printf语句之后进入wait等待，此时第二个子进程上台执行完printf之后执行exit (2) 终止，父进程从wait返回并回收第二个子进程的PCB，之后父进程结束，将未执行结束完毕的子进程一的父进程该为1号进程，T0+1000时刻，子进程1唤醒，执行printf语句之后执行exit (1) 终止，由1号进程回收其PCB。

子进程 009 的 PCB，父进程 007 回收。T0 时刻（之后的一小会）回收。

子进程 008 的 PCB，1#进程回收。T0+100s 回收。

（父进程 007 终止时，将未终止子进程的 ppid 改成 1，子进程 008 终止的时候，1#进程回收其 PCB）。

代码1.5 执行这个程序，系统需要使用几个进程？画与这个应用程序执行相关的进程树。

```
L1: #include <stdio.h>
L2: void main(void)
L3: {    int i;
L4:     printf ("%d  %d \n", getpid (),  getppid ());
L5:     for (i = 0; i < 3; ++i)
L6:         if ( fork() == 0 )
L7:             printf ("%d  %d \n", getpid (),  getppid ());
L8: }
```

系统需要10个进程（这是具体考虑到UNIX V6++系统后的结果），0号进程和1号进程为系统刚开始初始化时的进程，2号进程为当前主进程，2号进程执行for循环创建了3、4、5号子进程，3号子进程返回创建了6,7号子进程，4号子进程返回时创建了8号子进程，6号子进程返回时又创建了9号子进程。

进程树：

