

进程的创建和终止

同济大学计算机系 操作系统作业

2023-12-7

学号 2154312

姓名 郑博远

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 邓蓉 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。

父进程输出 2 个 20，子进程输出两个 10。

程序的输出：

可能是： 10 20 10 20
 （子进程输出 1 行，父进程输出 1 行，子进程再输出 1 行，父进程再输出 1 行）

可能是： 10 20 20 10
 （子进程输出 1 行，父进程输出 2 行，子进程再输出 1 行）

还可能是： 20 10 20 10
 （父进程输出 1 行，子进程输出 1 行，父进程再输出 1 行，子进程再输出 1 行）

或： 20 10 10 20
 （父进程输出 1 行，子进程输出 2 行，父进程再输出 1 行）

或： 20 20 10 10
 （父进程输出 2 行，子进程输出 2 行）

或： 10 10 20 20
 （子进程输出 2 行，父进程输出 2 行）

最后 2 种情况是常见的。

其余非常罕见。可能的原因是，这个应用程序是和其它应用程序并发执行的，进程有可能输出一条语句后因响应中断放弃 CPU。

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);
    }
}

```

答:

(1) 若父进程先被调度上台:

```

It is parent process. PID = 7, i = 8
It is child process. PID = 8, i = 0
The finished child process is 8.
The exit status is 256.

```

若子进程先被调度上台:

```

It is child process. PID = 8, i = 0
It is parent process. PID = 7, i = 8
The finished child process is 8.
The exit status is 256.

```

答案正确:)

补充: T0 时刻子进程终止, 父进程 007 回收子进程 008 的 PCB。

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。
- (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。

答：

(1)

It is parent process. PID = 7, i = 8

It is child process. PID = 8, i = 0 (100s 后)

(2) 子进程首先执行 `sleep` 系统调用入睡，放弃 CPU。100s 后，子进程被时钟中断唤醒，打印语句后调用 `exit` 系统调用终止。终止后，在 `exit` 系统调用中会唤醒正在 `wait` 入睡的父进程。但此时父进程已经终止 (早就终止了)，已经将所有子进程转交给 1#进程。因此，子进程的 PCB 在 `exit` 系统调用时唤醒 1#进程时，由 1#进程回收。

答案正确：)

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。
- (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。

答：

(1)

若父进程先被调度上台：

```
It is parent process. PID = 7, i = 8
It is child process. PID = 8, i = 0
```

若子进程先被调度上台：

```
It is child process. PID = 8, i = 0
It is parent process. PID = 7, i = 8
```

答案正确:)

(2) 无论父进程、子进程二者谁先被调度上台，父进程都会由于执行 `sleep` 系统调用而入睡放弃 CPU，因此子进程执行 `exit` 系统调用终止时父进程还未终止。子进程在 `exit` 系统调用中会唤醒正在 `wait` 入睡的父进程。但此时入睡的父进程是执行 `sleep` 系统调用入睡而非 `wait`，其 `p_wchan=&Time::tout` 而非其 PCB 的地址，此时不会被唤醒。待父进程被时钟中断唤醒，执行完毕终止时，会将子进程转交给 1#进程并将其唤醒。1#进程被调度上台后，便会回收其 PCB。答案正确:)

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。

(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。

答：

(1)

若在 fork 出第二个子进程后，父进程先被调度上台：

```
It is parent process. PID = 7, First Son: 8
It is parent process. PID = 7, Second Son: 9
It is child process. PID = 9, i = 0
Exit Son: 9. Exit Status= 512 2 (统一不左移了)
It is child process. PID = 8, i = 0
```

若在 fork 出第二个子进程后，该子进程先被调度上台：

```
It is parent process. PID = 7, First Son: 8
It is child process. PID = 9, i = 0
It is parent process. PID = 7, Second Son: 9
Exit Son: 9. Exit Status= 512 2 (统一不左移了)
It is child process. PID = 8, i = 0
```

(上述最后一行在 100s 后输出)

(3) 对于第二个子进程，需分两种情况讨论：

① fork 出该子进程后，父进程先被调度上台：

父进程打印 print 后，执行 wait 系统调用入睡。此时，第二个子进程是唯一的就绪进程，调度上台。打印 print 后，该子进程 exit 终止，唤醒因 wait 而入睡的父进程，由父进程回收其 PCB。

② fork 出该子进程后，其先被调度上台：

该子进程打印 print 语句后，exit 终止。此时父进程还未执行 wait 系统调用，因此不会唤醒父进程。待父进程上台后执行到 wait 系统调用时，发现了已经终止的第二个子进程，因此不会入睡，在此时回收其 PCB。之后系统调用返回，继续打印，然后终止。

啰嗦了点但应该是正确的，总之就是 T0 时刻附近父进程回收了。

对于第一个子进程 (008)，无论 fork 出它后父进程与其谁先调度上台，该子进程都会 sleep 入睡。T0+100s 时，待其被时钟中断唤醒时，父进程已经执行完毕终止，将其转移给 1#进程。因此，该子进程的 PCB 在 exit 系统调用时唤醒 1#进程时，由 1#进程回收。

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>


```

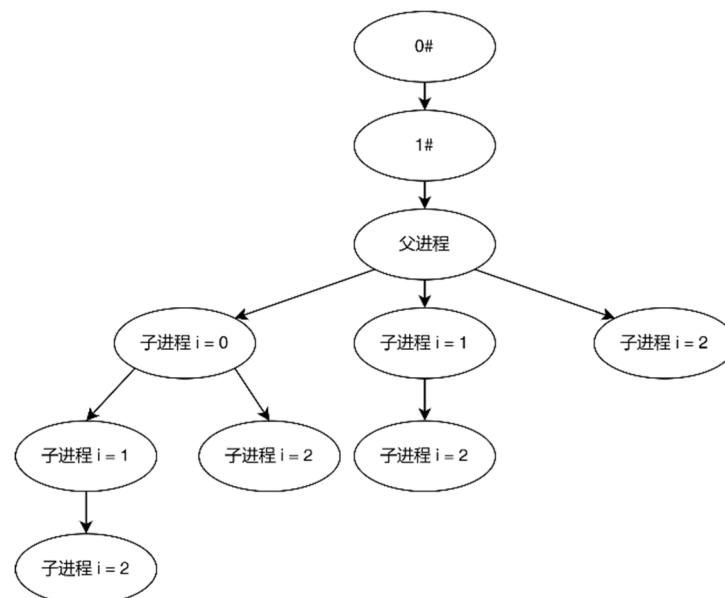
L2: void main(void)
L3: {    int i;
L4:     printf ("%d  %d \n", getpid (),  getppid ());
L5:     for (i = 0; i < 3; ++i)
L6:         if ( fork() == 0 )
L7:             printf ("%d  %d \n",  getpid (),  getppid ());
L8: }

```

答:

(1) 系统需要使用 10 个进程，分别是：0#，1#，父进程，fork 出的 7 个子进程。
(不算 0#，1#，那就是 8 个)

(2) 进程树:



注意，如果子进程打印前父进程终止了，那打印出来的 ppid 就是错的。

如果子进程输出前，父进程未终止：

p0 1 (shell 进程的 pid 号)

p1 p0

p2 p0

p3 p0

p11 p1

p12 p1

p21 p2

p111 p11

思考:如果不希望 ppid 是 1,那父进程就应该 wait。在上述程序中 else 分支加上 wait(),
就能让父进程等待子进程结束后再继续。