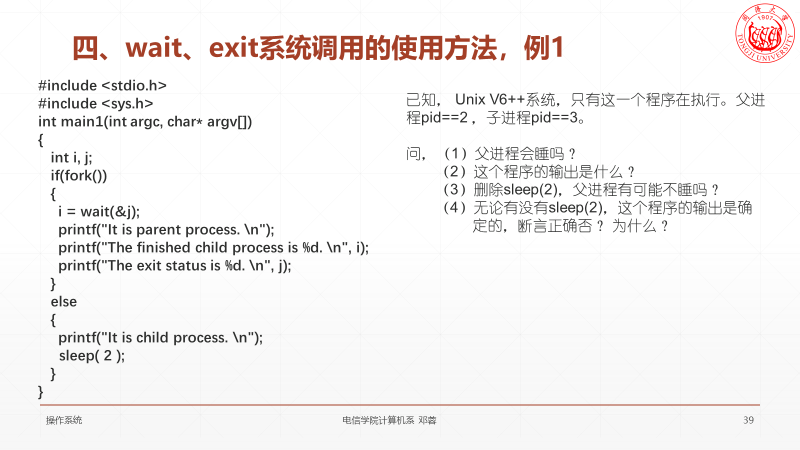
**进程的创建和终止**

同济大学计算机系 操作系统作业 2023-12-7

学号 姓名

1. 例1



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.

1. 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。

1. 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）

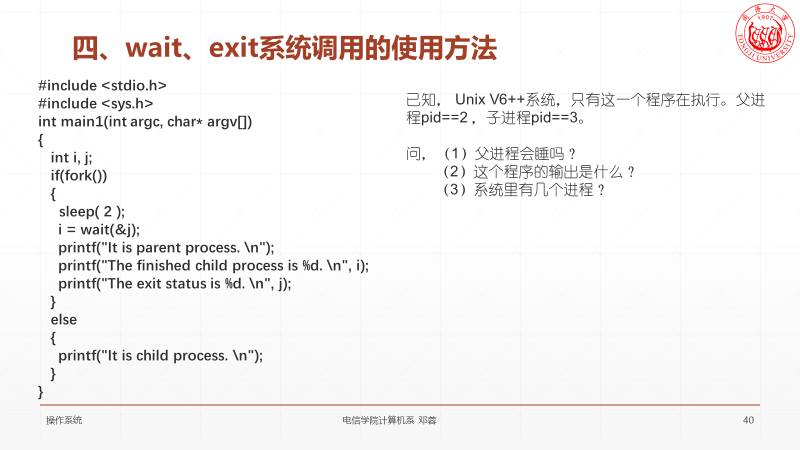
和终止码（0），赋给i变量和j变量。

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，

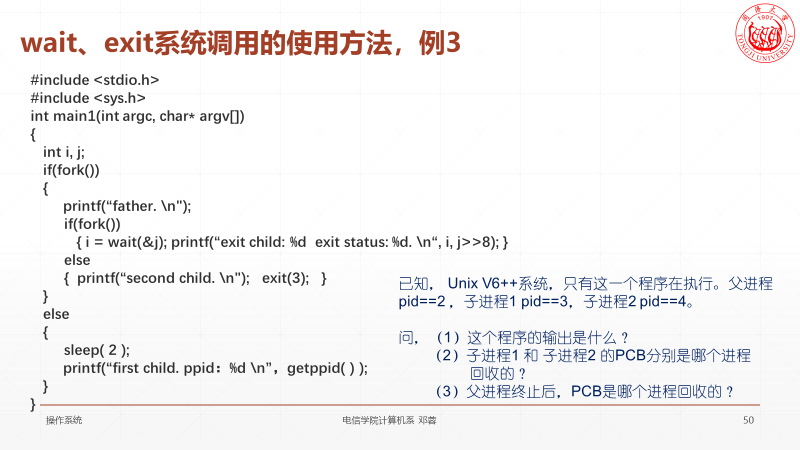
之后输出命令行提示符 #，等待用户输入下个命令行。\*\*\*\*\*\*程序执行，到此结束。

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。

1. 例2



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）。
4. 例3。有改动。



(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);

}

}

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。

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。

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。

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: }