实验三 Linux进程控制2

**学号： 20020006107 姓名： 王义钧 年级： 2020**

**一.使用wait系统调用**

**1.实验要求**

1. 编写一C/C++语言程序（程序名为testwait.c/testwait.cpp），使用系统调用wait()控制进程的阻塞。
2. 多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其原因。
3. 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码1所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出19os的式样。

**2.实验结果截图**

**结果分析：**

在父进程中，首先要创建一个子进程，然后sleep等待1秒，在这1s内，子进程可以执行自己的工作。

在子进程中，利用getpid()获取自己的进程号，然后结束子进程，并设置退出的参数为5。

父进程在sleep结束后，首先要执行wait()函数，由于子进程在1s内完成了对进程号的输出，并返回了子进程返回的结束值。在wait函数中，会把子进程退出时的状态取出并存入其中（status），这是一个整数值（int），指出了子进程是正常退出还是被非正常结束的，以及正常结束时的返回值PID。由于这些信息被存放在一个整数的不同二进制位中，所以用常规的方法读取会非常麻烦，故使用宏WEXITSTATUS()，我们可以用这个宏来提取子进程的返回值，如果子进程调用exit(5)退出，WEXITSTATUS(status)就会返回5。

**尝试情况：**

尝试在子进程中加入sleep，并使sleep时间长于父进程中的等待时间，可以看到输出的顺序发生了改变。

这是因为父进程在sleep结束后，子进程仍在sleep中，没有完成输出工作，故父进程调用wait()函数后阻塞，直到子进程完成为止，使用exit退出，父进程再次被唤醒。

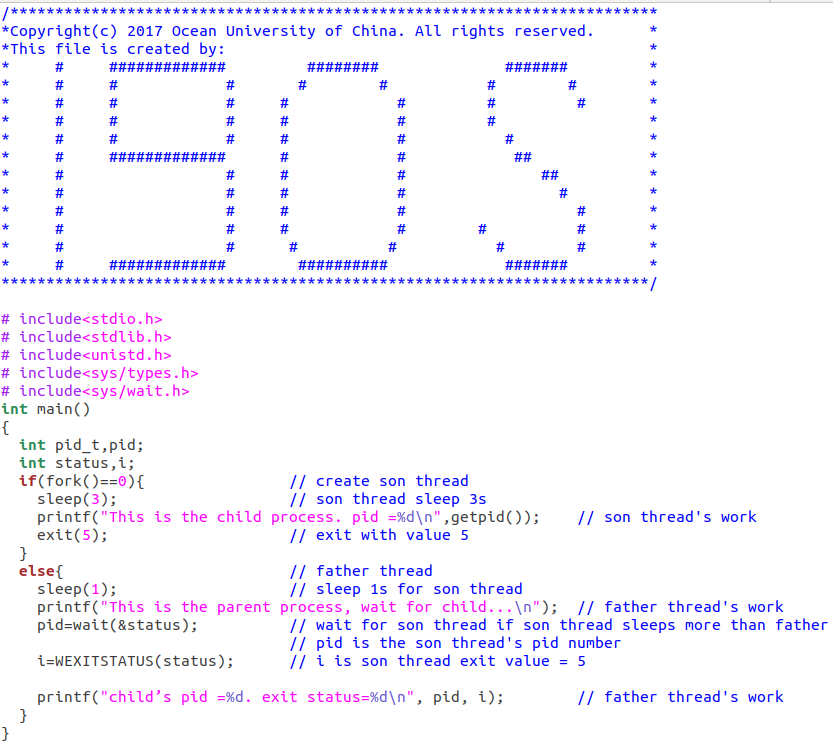


图 1 实验代码

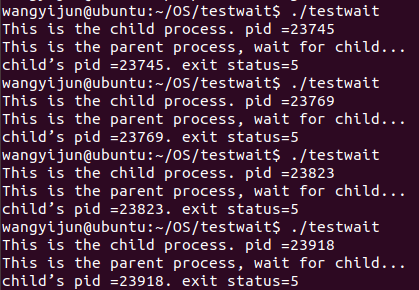


图 2 输出子进程的pid

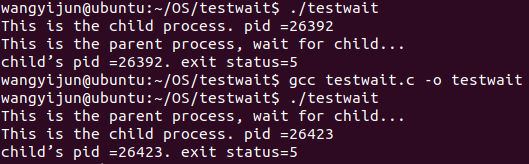


图 3 父进程的wait阻塞

**二.使用exit系统调用**

**1.实验要求**

1. 编写一C/C++语言程序（程序名为testexit.c/testexit.cpp），使用系统调用exit( )尝试对进程进行终止操作。
2. 多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其原因。
3. 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码1所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出19os的式样。

**2.实验结果截图**

**结果分析：**

父进程首先输出“this process will exit！”，然后利用exit结束掉当前进程，这样在后面的“never be displayed!”就不会显示。



图 4 实验代码

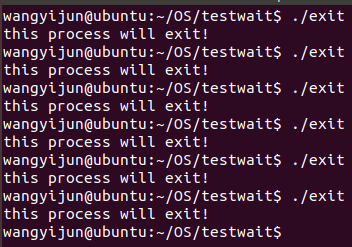


图 5 exit运行结果

**三.使用lockf系统调用**

**1.实验要求**

1）编写一C/C++语言程序（程序名为testlockf.c/testlockf.cpp），使用系统调 用lockf( )尝试对程序的指定区域进行加解锁操作。

2）多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其 原因。

3） 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释， 对代码关键逻辑步骤进行解释。在代码头部添加如代码1所示式样的头 部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添 加边框，并拼出自己汉字名字的式样。

**2.实验结果截图**

**结果分析：**

lockf(1,1,0)是锁定屏幕输出，不让其他进程可以输出到屏幕，lockf(1,0,0)则是解锁。利用系统调用lockf（fd，mode，size），对指定区域（有size指示）进行加锁或解锁，以实现进程的同步或互斥。其中，fd是文件描述字；mode是锁定方式，mode=1表示加锁，mode=0表示解锁；size是指定文件fd的指定区域，用0表示从当前位置到文件结尾。

首先，父进程先锁住了屏幕输出的资源，直到父进程的结束为止。故首先输出：Parent process: a。

然后在父进程中建立两个子进程，这两个子进程也在相互竞争抢夺处理机资源，就像实验二中的父子进程一样。在大多数情况下，b进程优先抢夺到资源，故执行：给屏幕输出上锁，输出自己是子进程1，pid是多少，process是b，然后给屏幕输出开锁。当b给屏幕上锁后，c无法获得屏幕资源，也就是说b，c两个子进程的处理过程可能有交叉，但只要一方执行了lockf(1,1,0)且没有开锁，那么另一方无法打印。

**尝试情况：**

假设在不上锁的情况下，当分别修改进程b和c中的打印次数为500次时，查看其输出情况。可以看到，在一开始大多是按照先b后c的顺序打印，但执行次数增大时，b与c的交叉执行现象越明显。

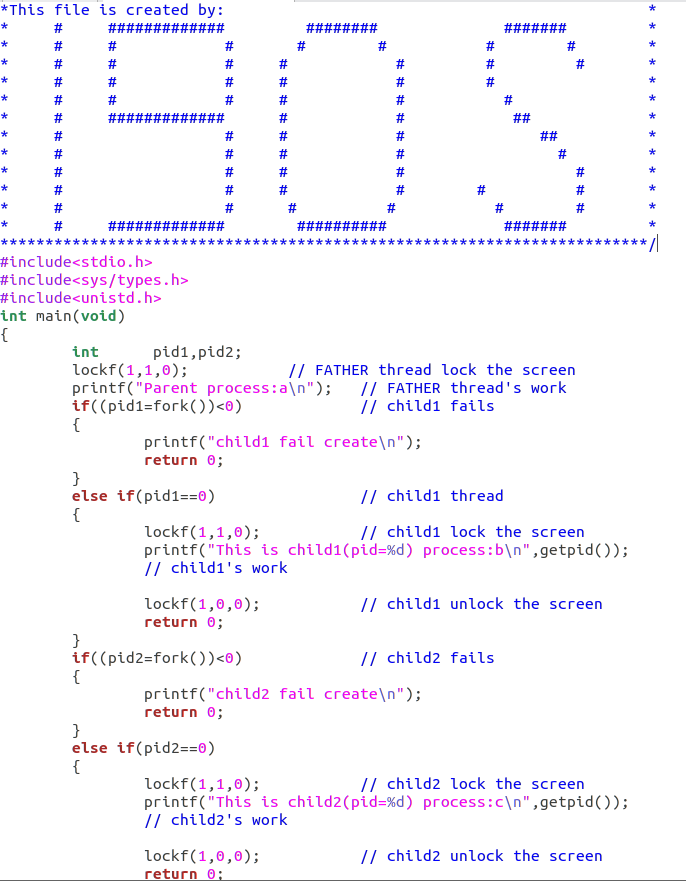


图 6 实验代码

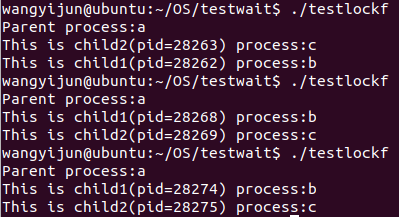


图 7 lockf运行代码



图 8 修改后的实验代码

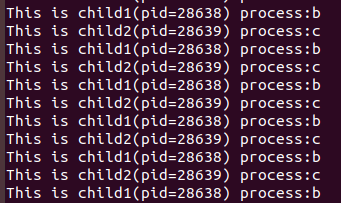


图 9 两个子进程的交叉进行