《操作系统》课程实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 进程的创建，软中断通信及进程的管道通信 | | | | | 实验序号 | 3 |
| 姓 名 | 吴伟玲 | 信息工程学院 | 信管 | 班 级 | 15 | 学 号 | 10207150468 |
| 实验日期 | 2012年12月11日 | | 指导教师 | 丘文峰 | | 成 绩 |  |
| **一、实验目的和要求**  加深对进程概念的理解，  明确进程和程序的区别  进一步认识并发执行的实质  分析进程争用资源的现象，学习解决进程互斥的方法  了解Linux系统中进程通信的基本原理 | | | | | | | |
| **二、实验预习内容**  1．实验指导书 | | | | | | | |
| **三、实验设备**  **配置名配置参数**  **硬盘 ≥8GB**  **内存 ≥1GB**  **磁盘文件格式NTFS, 以支持大的镜像文件操作系统Win32 XP +软件**  **VirtualBox Red Hat Linux Server ISO安装软件包** | | | | | | | |
| **四、实验内容及步骤**  1.0进程的创建编写一段程序，使用系统调用fork()创建两个子进程。当此程序运行时，在系统有一个父进程 和两个子进程活动。让每一个进程在屏幕上显示一个字符:父进程显示字符"a";子进程分别显示字 符"b"和字符"c"。试观察记录屏幕上的显示结果，并分析原因。  编写源程序create\_process.c  1.1 修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话，在观察程序执行时屏幕上 出现的现象，并分析原因。  修改源程序create\_process.c  1.2如果在程序中使用系统调用lockf()来给每一个进程加锁，可以实现进程之间的互斥，观察并分析 出现的现象。  修改源程序create\_process.c(系统调用lockf())  2.0. 使用系统调用fork()创建两个子进程，再用系统调用signal()让父进程捕捉键盘上来的中断 信号（即按Del键）;当捕捉到中断信号后，父进程用系统调用kill()向两个子进程发出信号，子进程捕捉 到信号后分别输出下列信息后终止：  child process 1 is killed by parent!  child process 2 is killed by parent!  父进程等待两个子进程终止后，输出如下的信息后终止：  parent process is killed!  编写源程序break1.c  2.1在上面的程序中增加语句signal(SIGINT, SIG\_IGN)和signal(SIGQUIT, SIG\_ING)观察执行结果，并分析原因  修改源程序break1.c  3.0编制一段程序，实现进程的管道通信。使用系统调用pipe()建立一条管道栈；两个子进程P1和P2分别向管道各写一句话：  child 1 is sending a message!  child 2 is sending a message!  而父进程则从管道中读出来自于两个子进程的信息，显示出屏幕上。 要求父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。 | | | | | | | |
| **五、实验结果、分析与心得**  1.0  输出结果为b,c,a.  原因分析：当启动系统时，存在父进程init，系统利用此父进程，调用fork( )创建子进程，函数返回进程id，id>0表示父进程，id=0表示子进程，id<0表示创建失败。  程序流程大概如下  (p1=fork())>0? //子进程p1已成功创建（转父进程？）  Yes:（（p2=fork())>0？//转父进程，子进程p2已成功创建（转父进程？）  Yes: print ‘a’ //(转入父进程执行)  No：print ‘c’ //(子进程2执行)  No :print ‘b’(子进程1执行)  当子进程成功创建后，父子进程间是相互独立的,理论上输出结果是不唯一的，b,c,a只是其中一种可能？  1.1  输出结果：child0~child49  50个son7242268  Daughter0~daughter49  分析：同1.0。  1.2程序加锁后执行结果为:  child0~child49。  原因分析：当子进程2成功创建后，并利用lockf(1,1,0)加锁，锁住整个子进程，直到子进程执行完毕，调用lockf(1,0,0)释放cpu资源。 为什么子进程2释放资源后其它的进程没有继续得到执行，并输出结果?  2.0  输出结果：p1,p2,parent  按ctrl+c后输出  parent process is killed！  结果分析：  由1.0可知，程序的执行流程应该如下  子进程P1被创建后，不转父进程，执行如下操作  Print ‘p1’ ;  Wait\_mark=1;  Signal( 16,stop) //接收父进程送来的信号，并转入预置的处理函数stop把wait\_mark置0,但此时父进程并未接收到用户信号，也无法向子进程发送信号，所以此语句不会有效。  Waiting( );  …… //因为wait\_mark=1，p1处于等待信号状态。  程序转入父进程，创建子进程p2,子进程p2创建后，执行p2  Print ‘p2’;  接着遇到的情况和p1相似……  程序转入父进程  Print ‘parent’;  Wait\_mark=1;  Signal(SIGINT,stop); //此时，当用户按下ctrl+c向父进程发送信号后，父进程转stop把wait\_mark置0。  Waiting( ); //已接收到信号，之间跳出循环。  Kill(p1,16);  P1: print ‘child process 1 is killed!’;  Kill(p2,16); //向子进程发送信号，此时程序流程应转到p1,p2?  P2: print ‘child process 2 is killed!’;  Wait(0); //同步。  Print ‘parent process is killed! ’;  Exit(0); //结束。  心得：按如上分析，输出结果应该为：  p1,p2,parent  ctrl+c: child process 1 is killed by parent!  child process 2 is killed by parent!  parent process is killed! 与实际输出结果不符。  2.1 程序在改动后的输出结果为：    分析与2.0相似，不同的是父进程中向子进程发送信号的操作置于中断预置处理函数intdelete( )中，子进程中的输出语句也放在了接受到信号后的预置处理含int1( )和int2（），同时，每一个进程都接收两个信号，SIGINT和SIGUSR(用户自定义信号)，两个信号被进程接收到后都转入相同的中断处理。  3.0输出结果为：    分析：系统调用pipe( )创建了一条管道，根据程序流程，p1调用write( )向管道发送一条信息“child 1 process is sending a message!” 父进程调用read( )读取管道信息，并输出在屏幕上，  然后p2执行和p1相同的操作后，父进程再次读取信息并输出。读取管道使用f[0],且只有父进程一个使用，不会发生资源竞争，不必lock,但写入管道时有p1和p2进程可能同时使用f[1]，所以在使用时调用了lock,以确保消息写入的准确性。 | | | | | | | |
| **教师评语：**  **成绩:**  **教师签字：** | | | | | | | |