山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100300063 | 姓名：李彦浩 | | 班级：软件菁英班 |
| 实验编号：实验*2* | | | |
| 实验题目：进程通信实验 | | | |
| 实验学时：2 | | 实验日期：2023.04.16 | |
| 实验目的：  通过 Linux 系统中管道通信机制，加深对于进程通信概念的理解，观察和体验 并发进程间的通信和协作的效果 ，练习利用无名管道进行进程通信的编程和调试技术 | | | |
| 硬件环境：  ThinkPad 8核 Intel Core i5-8265U 1.60GHz 8G内存  山大云算力平台 devlinux 1核 1G内存 | | | |
| 软件环境：  山大云算力平台 devlinux | | | |
| 实验步骤与内容：  独立实验要求根据用户给出的值建立三个并发进程，协作计算二元函数f(x,y)的，f(x)和f(y)的计算均服从一个递推式。    仿照示例程序给出的代码，建立两个管道pipe1和pipe2，一个进程计算完f(x)最终的结果后写入pipe1,另一个进程计算f(y)最终的结果后写入pipe2。然后第三个进程从两个管道中取出结果，计算最终的f(x,y)。  流程图如下。      利用之前学习的fork()函数，建立并发进程。第一个进程从管道1的1端写入，因此需要关闭pipe1[0]，然后循环计算出结果写入pipe1[1],然后关闭pipe1[1],退出进程。    如果运行的不是计算f(x)的函数，那么就要fork()另一个子进程用来计算f(y)，仅计算逻辑更改为f(y)的递推式。    随后，如果计算的不是f(x)也不是f(y)，那么计算f(x,y)的最终值。close掉两个管道的写入端，然后读取两个管道内前两个进程计算出的结果，相加得最终结果，然后关闭两个管道的写端，并退出。    在执行实验代码部分上，我采用了山东大学云计算平台提供的linux实例，编写了Makefile,将代码上传至实例中的数据域中。  在控制台中gmake，对代码进行编译。    然后在cd到这个文件夹下，执行./ppipe。程序会要求你输入两个值，一个x一个y，输入完就可以看到三个进程的计算过程。（含每一个递推式的迭代）    可以看到结果中，如果输入的x=5,y=5则f(x)=5\*4\*3\*2\*1,f(y)=5(斐波那契第5项)，然后和为125。  到这里整个独立实验要求的功能就都已经实现了。  本次实验实际投入学时2h,主要时间均用于代码的编写、调试与执行上  实验采用一种测试方式。输入x=y=5，得到结果为125（把具体计算过程也打印出来了）    实验中我创建了三个并发的进程来协作计算一个二元函数。每个进程计算完毕后就将计算结果放入对应的管道中，第三个进程通过管道和其他两个进程进行通信。  可以发现子进程中的局部变量原本是不可以被外部访问的，但通过管道通信机制，就可以用管道保存这个局部变量，使得其他进程可以从外部访问这个变量的值。因此管道本质上是个缓冲区，数据从写端流入，从读端流出。（有点像一个队列） | | | |
| 结论分析与体会：  实验中遇到了一个问题，如果直接同时利用fork创建子进程，相当于把整个单进程分裂成了四个进程，即子进程1，父进程1，子进程2，父进程2，结果就出现了偏差。因此需要先创建子进程1，父进程，然后考察是否是子进程，如果不是再fork一个子进程即可。  这样问题就解决了。  实验结果在上一个部分中已经列出，与预测的结果完全一致，完全复现了设计目标。  代码功能上还有可以增加的地方。在这个独立实验中，子进程1和子进程2之间是没有通信协作的。但如果我们更改一下递推式，就需要编写代码实现两个子进程之间的协作了。  比如，fx(t) = fy(t-1)\*fx(t-1),t>2;fx(1)=1;  fy(t) = fx(t)+fy(t-1),t>2;fy(1)=1;  这样如果要迭代计算，需要先计算fx(2) = fy(1)\*fx(1) = 1,然后计算fy(2) = fx(2) + fy(1) = 2,然后计算 fx(3) = fy(2) \* fx(2) = 2,…如此反复。这样就需要子进程1计算的时候读取pipe2的读端，子进程2计算的时候读取pipe1的读端等。  实验中我创建了三个并发的进程来协作计算一个二元函数。每个进程计算完毕后就将计算结果放入对应的管道中，第三个进程通过管道和其他两个进程进行通信。  可以发现子进程中的局部变量原本是不可以被外部访问的，但通过管道通信机制，就可以用管道保存这个局部变量，使得其他进程可以从外部访问这个变量的值。因此管道本质上是个缓冲区，数据从写端流入，从读端流出。（有点像一个队列） | | | |