山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100300157 | 姓名：孙荣骏 | | 班级：21.3 |
| 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：2022.4.21 | |
| 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：  宿主机  机型：联想拯救者r7000p2021  CPU：AMD R7 5800H  内存：16G  虚拟机  RAM：4GB | | | |
| 软件环境：  虚拟机：Ubuntu 16.04  宿主机：win10 | | | |
| 实验步骤与内容：  **独立实验：**    主要实验代码及注释如下：  /\*  \* Filename: os2.c  \* Copyright: 2023 srj  \* Date: 2023/04/21  \* Function: 建立三个并发协作进程，分别完成三个函数的实现。  \*/  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int fx (int);  int fy (int);  int main()  {  // 父进程处理f(x, Y)  int pid1; // 处理f(X)的子进程  int pid2; // 处理f(y)的子进程  int pipe1[2]; //　pipe1处理f(x)和父进程  int pipe2[2]; //　pipe2处理f(y)和父进程  int x, y; // 用户输入的数据x和y    // 获取用户输入需要计算的数据  printf("请输入整数x（x >= 1）：");  scanf("%d", &x);  // 检测输入x的合法性  while (x < 1)  {  printf("错误！请重新输入整数x（x >= 1）：");  scanf("%d", &x);  }    printf("请输入整数y（y >= 1）：");  scanf("%d", &y);  // 检测输入y的合法性  while (y < 1)  {  printf("错误！请重新输入整数y（y >= 1）：");  scanf("%d", &y);  }    // 事实上，应该将输入数据以字符数组的形式获得，再转成数字，进行判断  // 但这个不是重点，为了简单，只判断范围是否符合      if (pipe(pipe1) < 0)  {  perror("1号管道创建失败！\n");  exit(EXIT\_FAILURE);  }  if (pipe(pipe2) < 0)  {  perror("２号管道创建失败！\n");  exit(EXIT\_FAILURE);  }  printf("管道创建成功！\n");    pid1 = fork();  if (pid1 < 0)  {  // 建立1号子进程失败  printf("1号子进程创建失败！\n");  exit(EXIT\_FAILURE);  }  else if (pid1 > 0)  {  // 此时1号子进程已经创建成功，当前处在父进程控制下  // 创造2号子进程  // 在这里创建2号子进程的原因是防止在1号子进程中创建他的子进程  pid2 = fork();  if (pid2 < 0)  {  // 建立2号子进程失败  printf("2号子进程创建失败！\n");  exit(EXIT\_FAILURE);  }  else if (pid2 > 0)  printf("子进程创建成功！\n"); // 此时在父进程中  }    if (pid1 > 0 && pid2 > 0)  {  // 此时在父进程中  // 父进程仅接受来自2个子进程的数据  close(pipe1[1]);  close(pipe2[1]);  // 接收结果  int ansX, ansY;  read(pipe1[0], &ansX, sizeof(int));  close(pipe1[0]);  read(pipe2[0], &ansY, sizeof(int));  close(pipe2[0]);  int result = ansX + ansY;  printf("运行结果：f(x) = %d; f(y) = %d; f(x, y) = %d\n", ansX, ansY, result);    }    // 1号子进程处理f(x)  if (pid1 == 0)  {  // 1号子进程只打开1号通道的写入端  close(pipe1[0]);  close(pipe2[0]);  close(pipe2[1]);  int result = fx(x);  write(pipe1[1], &result, sizeof(int)); // 向父进程发送结果  close(pipe1[1]); // 结束后关闭通道    }    // 2号子进程处理f(y)  if (pid2 == 0 && pid1 > 0)  {  // 判断条件中加上"pid1 > 0"的原因是：1号子进程创建时所复制的数据pid2为0  // 因此在1号子进程中，如果不加以限制，会执行2号子进程的代码，这是不对的    // ２号子进程只打开２号通道的写入端  close(pipe2[0]);  close(pipe1[0]);  close(pipe1[1]);  int result = fy(y);  write(pipe2[1], &result, sizeof(int)); // 向父进程发送结果  close(pipe2[1]); // 结束后关闭通道  }    return EXIT\_SUCCESS;  }  // f(x)处理函数  int fx (int x)  {  if (x == 1)  return 1;  return x \* fx(x-1);  }  // f(y)处理函数  int fy (int y)  {  if (y == 1 || y == 2)  return 1;  return fy(y-1) + fy(y-2);  }  # srj  os2: os2.o  gcc os2.o -o os2  os2.o: os2.c  gcc -g -c os2.c  .PHONY: clean  clean:  rm os2 \*.o | | | |
| 结论分析与体会：  在本次试验中，我通过Linux 系统中管道通信机制，加深了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 ，并且练习利用管道进行进程通信的编程和调试技术。  通过查阅课外资料，我了解到了管道的原理和读写规则 | | | |