山东大学 软件 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100150209 | 姓名： 杨佳庆 | | 班级： 21级网安班 |
| 实验题目：实验二 进程通信实验 | | | |
| 实验学时：2小时 | | 实验日期：2023/4/27 | |
| 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：  设备名称 LAPTOP-UKNRNMI2  处理器 Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz  机带 RAM 8.00 GB (7.88 GB 可用)  设备 ID E6D4BD13-0B7C-4171-8511-F3F4F35DB823  产品 ID 00326-40000-00000-AAOEM  系统类型 64 位操作系统, 基于 x64 的处理器  笔和触控 笔支持 | | | |
| 软件环境：  山东大学公共计算云平台 | | | |
| 实验步骤与内容：  实验运行结果：        代码：  exp2.c:  /\*  \* Filename : exp2.c  \* copyright : (C) 2023 YJQ  \* Function : 建立三个并发协作进程，分别完成三个函数的实现。  \*/  #include <sys/types.h>  #include <unistd.h>  #include <signal.h>  #include <stdio.h>  #include <stdlib.h>  #include <wait.h>  #define M 10  int x[M],y[M];  int f1()  {      int i;      x[1] = 1;      for(i = 2; i < M; i++)      {          x[i] = x[i-1]\*i;      }  }  int f2()  {      int i;      y[1] = 1;      y[2] = 1;      for(i = 3; i < M; i++)      {          y[i] = y[i-1] + y[i-2];      }  }  int main(int argc, char \*argv[])  {      pid\_t pid1,pid2;   // 进程号      int pipe1[2];    // 存放第一个无名管道标号      int pipe2[2];    // 存放第二个无名管道标号      int m, n;         // 存放要传递的整数      int i,j;      f1();      f2();      do      {          printf("请输入整数m（m >= 1）：");          scanf("%d", &m);          if(m < 1) printf("错误！请重新输入");      }while (m < 1);        do      {          printf("请输入整数n（n >= 1）：");          scanf("%d", &n);          if(n < 1) printf("错误！请重新输入");      }while (n < 1);        // 使用 pipe()系统调用建立两个无名管道。建立不成功程序退出，执行终止      if (pipe(pipe1) < 0)      {          perror("pipe not create");          exit(EXIT\_FAILURE);      }      if (pipe(pipe2) < 0)      {          perror("pipe not create");          exit(EXIT\_FAILURE);      }      // 使用 fork()系统调用建立子进程,建立不成功程序退出，执行终止      if ((pid1 = fork()) < 0)      {          perror("process 1 not create");          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          if ((pid2 = fork()) < 0)          {              // 建立2号子进程失败              printf("process 2 not create\n");              exit(EXIT\_FAILURE);          }          else if (pid2 > 0)              printf("process create successfully\n");      }       // pid1 为 0 表示 1 号子进程在执行, 1 号子进程负责执行f(x)      if (pid1 == 0)      {          // 1号子进程只负责从管道 1 的 1 端写，          // 所以关掉管道 1 的 0 端和管道 2 的 0 端以及 1 端。          close(pipe1[0]);          close(pipe2[0]);          close(pipe2[1]);          // 每次循环从管道 1 的 0 端读一个整数放入变量 X 中,          // 并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10          printf("child %d read: %d write: %d\n", getpid(), m, x[m]);          write(pipe1[1], &x[m], sizeof(int));          // 读写完成后,关闭管道          close(pipe1[1]);          sleep(1);          // 子进程执行结束          exit(EXIT\_SUCCESS);      }      // pid2 为 0 表示 2 号子进程在执行, 2 号子进程负责执行f(y)      if (pid2 == 0 && pid1 > 0)      {          // 2号子进程只负责从管道 1 的 1 端写，          // 所以关掉管道 1 的 0 端和管道 2 的 0 端以及 1 端。          close(pipe2[0]);          close(pipe1[0]);          close(pipe1[1]);          // 每次循环从管道 2 的 0 端读一个整数放入变量 X 中,          // 并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10          printf("child %d read: %d write: %d\n", getpid(), n, y[n]);          write(pipe2[1], &y[n], sizeof(int));          // 读写完成后,关闭管道          close(pipe2[1]);          sleep(1);          // 子进程执行结束          exit(EXIT\_SUCCESS);      }      // 子进程号大于 0 表示父进程在执行,      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          waitpid(pid1, NULL, 0);          waitpid(pid2, NULL, 0);          close(pipe1[1]);          close(pipe2[1]);          // 接收结果          read(pipe1[0], &i, sizeof(int));          read(pipe2[0], &j, sizeof(int));          printf("parent %d read: %d %d \n", getpid(), i, j);          printf("f(%d,%d) = %d \n",i , j, i+j);          close(pipe1[0]);          close(pipe2[0]);      }      // 父进程执行结束      return EXIT\_SUCCESS;  }  Makefile:  exp2: exp2.o  gcc exp2.o -o exp2  exp2.o: exp2.c  gcc -g -c exp2.c  .PHONY: clean  clean:  rm exp2 \*.o  lab2.c：  /\*  \* Filename : exp2.c  \* copyright : (C) 2023 YJQ  \* Function : 建立三个并发协作进程，分别完成三个函数的实现。  \*/  #include <sys/types.h>  #include <unistd.h>  #include <signal.h>  #include <stdio.h>  #include <stdlib.h>  #include <wait.h>  #define M 10  int x[M],y[M];  int f1()  {      int i;      x[1] = 1;      for(i = 2; i < M; i++)      {          x[i] = x[i-1]\*i;      }  }  int f2()  {      int i;      y[1] = 1;      y[2] = 1;      for(i = 3; i < M; i++)      {          y[i] = y[i-1] + y[i-2];      }  }  int main(int argc, char \*argv[])  {      pid\_t pid1,pid2;   // 进程号      int pipe1[2];    // 存放第一个无名管道标号      int pipe2[2];    // 存放第二个无名管道标号      int m = 1, n = 1;         // 存放要传递的整数      int i,j;      f1();      f2();      for(;m<M;m++,n++){      // 使用 pipe()系统调用建立两个无名管道。建立不成功程序退出，执行终止      if (pipe(pipe1) < 0)      {          perror("pipe not create");          exit(EXIT\_FAILURE);      }      if (pipe(pipe2) < 0)      {          perror("pipe not create");          exit(EXIT\_FAILURE);      }      // 使用 fork()系统调用建立子进程,建立不成功程序退出，执行终止      if ((pid1 = fork()) < 0)      {          perror("process 1 not create");          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          if ((pid2 = fork()) < 0)          {              // 建立2号子进程失败              printf("process 2 not create\n");              exit(EXIT\_FAILURE);          }          else if (pid2 > 0)              printf("process create successfully\n");      }       // pid1 为 0 表示 1 号子进程在执行, 1 号子进程负责执行f(x)      if (pid1 == 0)      {          // 1号子进程只负责从管道 1 的 1 端写，          // 所以关掉管道 1 的 0 端和管道 2 的 0 端以及 1 端。          close(pipe1[0]);          close(pipe2[0]);          close(pipe2[1]);          // 每次循环从管道 1 的 0 端读一个整数放入变量 X 中,          // 并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10          printf("child %d read: %d write: %d\n", getpid(), m, x[m]);          write(pipe1[1], &x[m], sizeof(int));          // 读写完成后,关闭管道          close(pipe1[1]);          sleep(1);          // 子进程执行结束          exit(EXIT\_SUCCESS);      }      // pid2 为 0 表示 2 号子进程在执行, 2 号子进程负责执行f(y)      if (pid2 == 0 && pid1 > 0)      {          // 2号子进程只负责从管道 1 的 1 端写，          // 所以关掉管道 1 的 0 端和管道 2 的 0 端以及 1 端。          close(pipe2[0]);          close(pipe1[0]);          close(pipe1[1]);          // 每次循环从管道 2 的 0 端读一个整数放入变量 X 中,          // 并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10          printf("child %d read: %d write: %d\n", getpid(), n, y[n]);          write(pipe2[1], &y[n], sizeof(int));          // 读写完成后,关闭管道          close(pipe2[1]);          sleep(1);          // 子进程执行结束          exit(EXIT\_SUCCESS);      }      // 子进程号大于 0 表示父进程在执行,      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          waitpid(pid1, NULL, 0);          waitpid(pid2, NULL, 0);          close(pipe1[1]);          close(pipe2[1]);          // 接收结果          read(pipe1[0], &i, sizeof(int));          read(pipe2[0], &j, sizeof(int));          printf("parent %d read: %d %d \n", getpid(), i, j);          printf("f(%d,%d) = %d \n",i , j, i+j);          close(pipe1[0]);          close(pipe2[0]);      }      }      // 父进程执行结束      return EXIT\_SUCCESS;  }  Makefile：  srcs = lab2.c  objs = lab2.o  opts = -g -c  all: lab2  lab2: $(objs)  gcc $(objs) -o lab2  lab2.o: $(srcs)  gcc $(opts) $(srcs)  clean:  rm lab2 \*.o | | | |
| 结论分析与体会：  结论分析：  管道是半双工的，管道单独构成一种文件系统，并且只存在与内存中。这种感觉，就像是一个丰巢快递柜一样，快递员把货送到柜子中，只有柜子中的货物被取走，该柜子才能放下一个货物。  体会：  通过上次实验，对fork()的使用，以及父子进程的执行熟悉了很多，因此这次实验用的时间不是很多，代码写的也非常舒服。出了２次错，一次是scanf()使用错误，并且没加分号，还有一个错误是数字输入问题，这一点挺奇怪的，数字的输入格式不对，Ubuntu中，中文数字和英文数字输入不同，但在Windows中是正常的，或许是我设置的问题。pipe()函数的理解，也是看各种博客和资料有了清晰的认识，总之，我们学习阶段，一定要充分利用网络资源，加深学习。  补充：运行结果中的“运算完毕，结果已发送”两次运行出现的顺序不同。两个子进程在打印语句后，都应该加上kill(getppid(), SIGCONT)信号，在父进程中输出结果前应该加上2句pause()。SIGNAL语句前面也加上，头文件补充一下。这样输出就可以按照顺序输出了。 | | | |