山东大学 计算机科学与技术 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900161140 | 姓名：张文浩 | | 班级： 19智能 |
| 实验题目：线程和管道通信实验 | | | |
| 实验学时：2 | | 实验日期： 4.12 | |
| 实验目的：  通过 Linux 系统中线程和管道通信机制的实验，加深对于线程控制和管道通信概念的理解，观察和体验并发进/线程间的通信和协作的效果 ，练习利用无名管道进行进/线程通信的编程和调试技术。 | | | |
| 硬件环境： 自带笔记本 | | | |
| 软件环境：vmware/ubuntu | | | |
| 实验步骤与内容：   1. 先根据两个实例程序学习线程管道通信和进程管道通信。以下分别是实例实验一和实例实验二的运行结果。   1  2   1. 进行独立实验，我分别通过建立三个并发的线程和三个并发的进程实验实验要求。 2. 并发线程实验思路为：创建三个线程thrd1，thrd2，thrd3。创建两个管道pipe1和pipe2。其中thrd1执行task1，即计算f（x），计算结果数据通过管道pipe1写到线程3（thrd3）。而另一个thrd2线程执行task2，即计算f（y），并将计算结果通过管道pipe2发送到线程3（thrd3）。最后在线程三（thrd3）中读入前两个线程的运算结果，并将结果相加，最后输出结果。具体代码如下   #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<pthread.h>  int pipe1[2],pipe2[2]; //record pip number  pthread\_t thrd1,thrd2,thrd3; //record three pthread number  void task1(int \*x){ //calculate f(x)  printf("task1 is running!\n");  int ansx=1;  for(int i=1;i<=\*x;i++){  ansx\*=i;  }  write(pipe1[1],&ansx,sizeof(int));  printf("thread1 write f(x) = %d to thread3\n",ansx);  close(pipe1[1]);  }  int f(int y){  if(y==1||y==2){  return 1;  }else{  return f(y-1)+f(y-2);  }  }  void task2(int \*y){ //calculate f(y)  printf("task2 is running!\n");  int ansy=f(\*y);  write(pipe2[1],&ansy,sizeof(int));  printf("thread2 write f(y) = %d to thread3\n",ansy);  close(pipe2[1]);  }  void task3(int \*num){ //calculate f(x,y)  printf("task3 is running!\n");  int ansx,ansy,ans;  read(pipe1[0],&ansx,sizeof(int));  printf("thread3 read f(x) = %d from thread1\n",ansx);  read(pipe2[0],&ansy,sizeof(int));  printf("thread3 read f(y) = %d from thread2\n",ansy);  close(pipe1[0]);  close(pipe2[0]);  ans=ansx+ansy;  printf("f(x,y) = %d\n", ans);    }  int main(int argc,char \*arg[])  {  int ret;  int x,y;  //create two pipe  if(pipe(pipe1)<0){  perror("pipe1 not create!");  exit(EXIT\_FAILURE);  }  if(pipe(pipe2)<0){  perror("pipe2 not create!");  exit(EXIT\_FAILURE);  }  //input x and y  printf("please input x,y");  scanf("%d%d",&x,&y);  //create three pthread by function pthread\_create  ret=pthread\_create(&thrd1,NULL,(void\*)task1,(void\*)&x);  if(ret){  perror("pthread\_create:f1 failire!");  exit(EXIT\_FAILURE);  }  ret=pthread\_create(&thrd2,NULL,(void\*)task2,(void\*)&y);  if(ret){  perror("pthread\_create:f1 failure!");  exit(EXIT\_FAILURE);  }  int num=1;  ret=pthread\_create(&thrd3,NULL,(void\*)task3,(void\*)&num);  if(ret){  perror("pthread\_create:f1 failure!");  exit(EXIT\_FAILURE);  }  //to clear three pthread  pthread\_join(thrd1,NULL);  pthread\_join(thrd2,NULL);  pthread\_join(thrd3,NULL);  //exit  exit(EXIT\_SUCCESS);  }  运行结果如下：  3   1. 并发进程实验思路为：创建四个管道pipe，和两个子进程，在第一个子进程里先通过管道一（pipe1）从父进程中读入x值，并在第一个子进程中算出f（x）的结果，并通过管道二（pipe2）将结果发回到父进程。在第二个子进程中先通过管道三（pipe3）从父进程中读入y值，并在第二个子进程中计算出f（y）的结果，并通过管道四（pipe4）将结果发回到父进程。最后在父进程中读取两个子进程发来的结果，进行相加运算并输出。   注：在每一个进程中会关掉没用的管道，在使用完管道后，因为管道只会使用到一次，所以也关掉刚刚用过的管道。具体代码如下：  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  int main()  {  int pid1,pid2; //the name of process  int pipe1[2],pipe2[2],pipe3[2],pipe4[2]; //create four pipe  //start creating  if(pipe(pipe1)<0){  perror("creating pipe1 failed!\n");  exit(EXIT\_FAILURE);  }  if(pipe(pipe2)<0){  perror("creating pipe2 failed!\n");  exit(EXIT\_FAILURE);  }  if(pipe(pipe3)<0){  perror("creating pipe3 failed!\n");  exit(EXIT\_FAILURE);  }  if(pipe(pipe4)<0){  perror("creating pipe4 failed!\n");  exit(EXIT\_FAILURE);  }  //create child process  pid1=fork();  if(pid1<0){  perror("creating child process1 failed!\n");  exit(EXIT\_FAILURE);  }  if(pid1==0){ //child process running  //close useless pipe  close(pipe1[1]);  close(pipe2[0]);  int x,ansx=1;  read(pipe1[0],&x,sizeof(int));  printf("child process:%d read x = %d from father process:%d\n", getpid(),x,getppid());  //calculate f(x)  for(int i=1;i<=x;i++){  ansx\*=i;  }  printf("child process:%d write f(x) = %d to father process:%d\n",getpid(),ansx,getppid());  write(pipe2[1],&ansx,sizeof(int));  //close used pipe  close(pipe1[0]);  close(pipe2[1]);  exit(EXIT\_SUCCESS);  }  if(pid1>0){  pid2=fork();  if(pid2<0){  perror("creating child process2 failed!\n");  exit(EXIT\_FAILURE);  }  if(pid2==0){  //close useless pipe  close(pipe3[1]);  close(pipe4[0]);  int y,ansy;  read(pipe3[0],&y,sizeof(int));  printf("child process:%d read y = %d from father process:%d\n", getpid(),y,getppid());  //calculate f(y)  int temp1=1,temp2=1;  if(y==1||y==2){  ansy=1;  }else{  for(int i=3;i<=y;i++){  ansy=temp1+temp2;  temp1=temp2;  temp2=ansy;  }  }  printf("child process:%d write f(y) = %d to father process:%d\n",getpid(),ansy,getppid());  write(pipe4[1],&ansy,sizeof(int));  //close used pipe  close(pipe3[0]);  close(pipe4[1]);  exit(EXIT\_SUCCESS);  }  if(pid2>0){  //close useless pipe  close(pipe1[0]);  close(pipe2[1]);  close(pipe3[0]);  close(pipe4[1]);  int x,y;  printf("please input x and y\n");  scanf("%d %d",&x,&y);  int ansx,ansy;  write(pipe1[1],&x,sizeof(int));  write(pipe3[1],&y,sizeof(int));  read(pipe2[0],&ansx,sizeof(int));  printf("father process read f(x) = %d\n",ansx);  read(pipe4[0],&ansy,sizeof(int));  printf("father process read f(y) = %d\n",ansy);  int ans=ansx+ansy;  printf("f(%d,%d) = %d\n", x,y,ans);  //close used pipe  close(pipe1[1]);  close(pipe2[0]);  close(pipe3[1]);  close(pipe4[0]);  exit(EXIT\_SUCCESS);  }  }  }  运行结果如下：  4 | | | |
| 结论分析与体会：  1.管道通信时消息传递的一种方式，所谓管道，是指用与练几个一个读进程和写进程来实现他们之间通信的一个共享文件。 管道机制必须提供以下三个方面的协调能力，互斥，同步和确定对方存在。  管道是单方向通信，要实现双方通信，必须实现两个管道。  2. 管道机制为进程之间的协作和消息传递提供一种很好的方法。  管道是半双工通信，给另一进程可以在管道的另一端从管道得到其输入。管道以半双工方式工作,即它的数据流是单方向的。因此使用一个管道一般的规则是读管道数据的进程关闭管道写入端,而写管道进程关闭其读出端。  3. 管道pipe 是进程间通信最基本的一种机制,两个进程可以通过管道一个在管道一端向管道发送其输出,给另一进程可以在管道的另一端从管道得到其输入.管道以半双工方式工作,即它的数据流是单方向的.因此使用一个管道一般的规则是读管道数据的进程关闭管道写入端,而写管道进程关闭其读出端，所谓管道，是指用与练几个一个读进程和写进程来实现他们之间通信的一个共享文件。  4. 两个进程可以通过管道一个在管道 一端向管道发送其输出,给另一进程可以在管道的另一端从管道得到其输入.管道以半双工方式工作,即它的数据流是单方向的.因此使用一个管道一般的规则是读管道 数据的进程关闭管道写入端,而写管道进程关闭其读出端。 | | | |