**《操作系统》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2017级计算机科学与技术专业卓越班** | | | **姓名** | **江焰丰** |
| **学号** | **20174261** |
| **实验题目** | **多进程和多线程** | | | | | |
| **实验时间** | **2019.10.08** | | **实验地点** | **A主0414** | | |
| **实验成绩** |  | | **实验性质** | **□综合性 □验证性□设计性** | | |
| 教师评价：  **□**算法/实验过程正确；  **□**源程序/实验内容提交  **□**程序结构/实验步骤合理；  **□**实验结果正确；  **□**语法、语义正确；  **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 1. 实验目的   1.加深对进程概念的理解，明确进程与线程的区别。  2.掌握Linux进程创建和撤销的方法，进一步认识并发执行的实质  3.了解多线程的程序设计方法。 | | | | | | |
| 二、实验项目内容  在Linux环境下用C语言编写程序，创建多个进程，基本实现以下内容：  1号父进程，创建3、4号进程。  3号进程实现对系统命令的调用执行，例如ls，ps，free等。  4号进程创建5、6号进程，用于完成以下任务：  5号进程创建两个线程Thread1，Thread2。其中  Thread1：求（1~n）之间的素数。  Thread2：生成Fibonacci序列。  6号进程执行一个用户写的可执行文件：输出hello,world！ | | | | | | |
| 三.试验过程及算法  **Step 1** .首先1号进程使用fork函数创建3号子进程和4号子进程。在创建完3号子进程后，调用wait函数，使4号进程等待3号进程执行完后再执行，避免混乱。最后，1号父进程使用waitpid函数等待3号子进程和4号子进程执行完后再继续执行。    **Step 2.** 3号进程进入childfunc函数内调用system函数执行ls,ps,free命令。    **Step 3.** 4号进程进入childfunc函数内继续调用Createprocess\_two函数，即创建5号子进程和6号子进程。同时调用wait函数使6号进程在5号进程执行之后再执行。      **Step 4.** 5号进程进入funct函数，再调用thread\_two函数使用pthread\_create函数创建两个线程。一个线程执行计算1-n范围内素数的数量，算法为对于每个数m,从2到m/2,判断能整除，如果能整除（除数与被除数不想等的情况下），那说明其不是素数，反之则为素数。另一个进程为计算n个斐波那切数列的值。具体为从第三项开始使用递推式，每一项的值为前两项之和。          **Step 5.** 6号进程使用system函数调用hello.c的可执行文件hello来打印Hello, world!。    **Step 6.** 最后为各个进程的退出。 | | | | | | |
| 四 实验中遇到的问题   * 1. 使用for循环创建进程会导致所创建的子进程在之后执行for循环时，也会创建子进程，导致进程数量增多和关系混乱。   2. 使用execlp函数时，在调用完第一个execlp函数后，后面的代码将不会执行。   3. 两个子进程之间未使用wait函数，导致执行顺序混乱。   具体解决方案在第七点， 实验心得中写出。  五 实验结果以及分析：    具体分析见第七点，实验心得。  六 实验程序代码：  #include<unistd.h>  #include<pthread.h>  #include<stdio.h>  #include<math.h>  #include<sys/types.h>  #include<sys/wait.h>  #define max\_len 500  *//primer number*  void \*MyThread1(void)  {      int number;      printf("now is the thread 1\n");      printf("please enter a number bigger than 1\n");      scanf("%d", &number);      int count = 1;      printf("1 ");      for(int  i=2;i<=number;i++)      {          int flag = 0;          for(int j=2;j<number/2;j++)          {              if(i % j == 0 && i != j)              {                  flag = 1;                  break;              }          }          if(flag == 0)          {              count++;              printf("%d ", i);              if(count % 5 == 0)                  printf("\n");          }      }      printf("\n");  }  *//Fibonacci  sequence*  void \*MyThread2(void)  {      int len;      printf("now is the thread 2\n");      printf("please enter the sequence len bigger than 2\n");      scanf("%d", &len);      int  fib[max\_len];      fib[0] = 1;      fib[1] = 1;      printf("1 1 ");      for(int i=2;i<len;i++)      {          fib[i] = fib[i-1] + fib[i-2];          printf("%d ", fib[i]);          if(i % 5 == 0)              printf("\n");      }      printf("\n");  }  int thread\_two()  {      pthread\_t  id1, id2;      int err1, err2;      err1 = pthread\_create(&id1, NULL, MyThread1, NULL);      if(err1 != 0)      {          printf("thread 1 can't create\n");          return 0;      }      pthread\_join(id1, NULL);      err2 = pthread\_create(&id2, NULL, MyThread2, NULL);      if(err2 != 0)      {          printf("thread 2 can't create\n");          return 0;      }      pthread\_join(id2, NULL);      printf("main thread exit!\n");      return 1;  }  void Createprocess\_two()  {      int i ;      pid\_t p5, p6;      p5 = fork();      if(p5 == 0)      {          funct(5);          return 0;      }      wait(NULL);      p6 = fork();      if(p6 == 0)      {          funct(6);          return 0;      }      int st1, st2;      waitpid(p5, &st1, 0);      waitpid(p6, &st2, 0);  }  void funct(int i)  {      int flag = 0;      switch (i)      {      case 5:          {              printf("now is process no.5\n");              flag = thread\_two();*// create the two thread*              if(flag == 0)                  printf("Error in process 5\n");              flag = 0;              break;          }      case 6:          {              printf("now is process no.6\n");              system("./hello");              break;          }      default:          break;      }  }  void childfunc(int i)  {      int flag = 0;      switch(i)      {  *//system call ls, ps and free*          case 3:              {                  printf("now is process 3\n");                  printf("now is ls command\n");                  system("ls");                  printf("now is ps command\n");                  system("ps");                  printf("now is free command\n");                  system("free");                  break;              }  *//no.4 process create the no.5 process and no.6 process*          case 4:              {                  printf("now is process 4\n");                  Createprocess\_two();                  break;              }          default:              break;      }  }  *//no.1 process create the no.3 process and no.4 process*  int Createprocess()  {      int i ;      int status;        pid\_t p3, p4;      p3 = fork();      if(p3 == 0)      {          childfunc(3);          return 0;      }      wait(NULL);      p4 = fork();      if(p4 == 0)      {          childfunc(4);          return 0;      }      int st1, st2;      waitpid(p3, &st1, 0); *//no.1 parent's process waits the child  process no.3*      waitpid(p4, &st2, 0);*//no.1 parent's process waits the child  process no.4*      return 0;  }  int main()  {      Createprocess();      return 0;  }  七 实验心得（实验结果及分析和（或）源程序调试过程（包含程序使用方法、程序运行截图），实验过程中遇到的问题分析与心得体会。（实验报告中最重要的部分，应尽量详细，重点描述自己遇到的问题以及解决方法））  **1.实验遇到的问题分析与解决**  a.使用for循环创建进程会导致所创建的子进程在之后执行for循环时，也会创建子进程，导致进程数量增多和关系混乱。  这个问题的解决方法之一是单独创建进程，而不采用for循环。当一个进程被fork出来后，判断是否为子进程，如果是直接调用进程需要执行的函数。  b.使用execlp函数时，在调用完第一个execlp函数后，后面的代码将不会执行。  改用system函数来调用系统命令，实现了连续调用ls, ps, free命令。  c.两个子进程之间未使用wait函数，导致执行顺序混乱。  在创建一个子进程后，如想再创建一个进程，为保证顺序一致，在创建下一个进程前调用wait函数，等上一个进程执行完毕后再创建。  **2.实验结果分析**    运行结果表面，1号父进程创建了3号子进程，3号子进程调用了ls,ps,free命令。之后是4号进程，其又创建了5号进程和6号进程。首先执行5号进程，5号进程创建了两个线程，一个用于计算一定范围内素数的数量，另一个计算出一定项数的斐波那契数列。最后6号进程调用hello.c的可执行文件hello，输出Hello, world!。  **3.实验心得**  本次实验让我对多进程和多线程加深了理解，同时学习到一些关于在Linux上进行进程，线程创建和管理的基础。 | | | | | | |