**广州大学学生实验报告**

开课学院及实验室: 计算机科学与网络工程学院 2024年4月26日

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | 计算机科学与网络工程学院 | 年级/专业/班 |  | 姓名 |  | 学号 |  |
| 实验课程名称 | 操作系统实验 | | | | | 成绩 |  |
| 实验项目名称 | 进程管理与进程通信 | | | | | 指导老师 |  |

**实验一 进程管理与进程通信**

1. **实验目的**

1、掌握进程的概念，明确进程的含义。

2、认识并了解进程并发执行的实质，进程的阻塞与唤醒，终止与退出的过程。

3、熟悉进程的睡眠、同步、撤消等进程控制方法。

4、分析进程竞争资源的现象，学习解决进程互斥的方法 。

5、了解什么是信号，利用信号量机制熟悉系统中进程之间软中断通信的基本原理，

6、熟悉消息传送的机理 ，共享存储机制 。

**二、实验软件**

Linux操作系统

VMware

GNU c++编译器

**三、实验内容：**

**1、编写一段程序，使用系统调用fork( )创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程并发执行，观察实验结果并分析原因。**

#include<iostream>

#include<stdio.h>

#include<unistd.h>

using namespace std;

int main()

{

    int p1,p2;

    while((p1 = fork())==-1);//创建子进程p1，创建成功向p1返回0

    if(p1 == 0)

    {

        putchar('b');

        cout<<"->该进程是子进程";

        cout<<endl;

    }

    else

    {

        while((p2 = fork())==-1);//创建子进程p2，创建成功向p2返回0

        if(p2==0)

        {

            putchar('c');

            cout<<"->该进程是子进程";

            cout<<endl;

        }

        else

        {

            putchar('a');

            cout<<"->该进程是父进程";

            cout<<endl;

        }

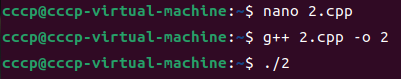
        return 0;

    }

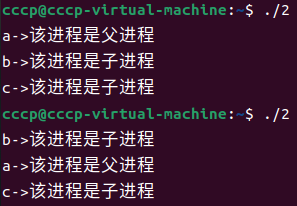
}

**结果显示:**

**编译:**

****

**运行结果:**

****

**思考题**

**（1）系统是怎样创建进程的？**

**1.申请一个空闲的PCB，为其分配一个空闲的PID**

**2.为进程分配空间**

**3.初始化新进程的PCB，包含进程的状态、标识符等**

**4.将新进程加入就绪队列**

**（2）当首次调用新创建进程时，其入口在哪里？**

**在进程队列的就绪状态下，由离自己最近的父进程执行调度，即入口在最近的父进程处。**

**（3）程序的多次运行结果为什么不同？如何控制实验结果的随机性？**

**这个代码有6种结果，分别是abc/acb/bac/bca/cab/cba。在语句执行fork()之前只有一个父进程在执行后续代码，每执行一次fork()后就会多出一个子进程来与父进程并发执行后续代码(子进程的开始在fork()完成后)。多条进程会同时争抢处理器资源，因此造成了有多种输出结果。可以使用wait()、exit()实现进程同步，控制进程的随意性。**

**2、用fork( )创建一个进程，再调用exec( )，用新的程序替换该子进程的内容，利用wait( )来控制进程执行顺序，掌握进程的睡眠、同步、撤消等进程控制方法，并根据实验结果分析原因。**

#include<stdio.h>

#include<unistd.h>

#include<iostream>

#include<sys/wait.h>

#include<stdlib.h>

using namespace std;

int main()

{

    int pid;

    pid = fork();

    switch(pid)

    {

        case -1:    //子进程创建失败

            printf("fork fail!\n");

            exit(1);

        case 0:     //子进程

            execl("/bin/ls","ls","-1","-color",NULL);//用新的进程替换子进程

            printf("exec fail!\n");//替换失败

            exit(1);

        default:    //父进程

            wait(NULL);//让父进程睡眠，等待子进程结束后将父进程唤醒

            printf("Is completed!\n");

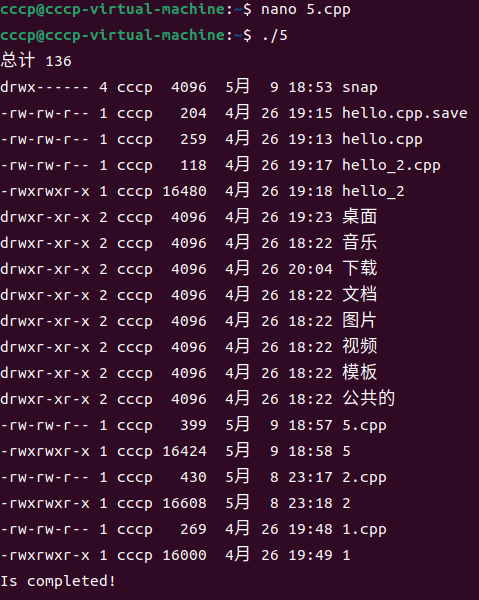
            exit(0);

    }

    return 0;

}

**运行结果:**

****

**思考题：**

**（1）可执行文件加载时进行了哪些处理？**

**加载器将可执行目标文件中的代码和数据从磁盘复制到内存中，然后通过跳转到程序的第一条指令或入口点来运行该程序。**

**（2）什么是进程同步？wait( )是如何实现进程同步的？**

**进程同步的主要任务是使并发执行的诸进程之间能有效地共享资源和相互合作，使执行的结果具有可再现性。**

**wait()可以让某一进程睡眠，等待另一进程完成后再执行该进程，从而实现进程的同步。**

**（3）wait( )和exit（）是如何控制实验结果的随机性的？**

**以以上代码为例，当创建一个子进程后，父进程与子进程同步运行。当执行父进程后，使用wait()将父进程睡眠，等待。在子进程结束后，执行exit(1),exit(1)执行后会终结子程序，并返回一个状态值给父进程，这个状态值能够唤醒父进程。**

**总而言之:wait(NULL)可以让进程睡眠，exit(1)可以唤醒父进程。**

**3、编写一段多进程并发运行的程序，用lockf( )来给每一个进程加锁，以实现进程之间的互斥，观察并分析出现的现象及原因。**

#include <stdio.h>

#include <unistd.h>

#include<iostream>

using namespace std;

int main()

{

   int pid1, pid2, i;

   pid1 = fork();

   if(!pid1)

   {

      lockf(1, 1, 0);//加锁

      for(i = 0; i < 4; i++)

      {

        printf("daughter-%d\n", i);

      }

      lockf(1, 0, 0);//解锁

   }

   else

   {

      pid2 = fork();

      if(!pid2)

      {

        lockf(1, 1, 0);//加锁

        for(i = 0; i < 4; i++)

        {

            printf("son-%d\n", i);

        }

        lockf(1, 0, 0);//解锁

      }

      else

      {

        lockf(1, 1, 0);//加锁

        for(i = 0; i < 4; i++)

        {

            printf("parent-%d\n", i);

        }

        lockf(1, 0, 0);//解锁

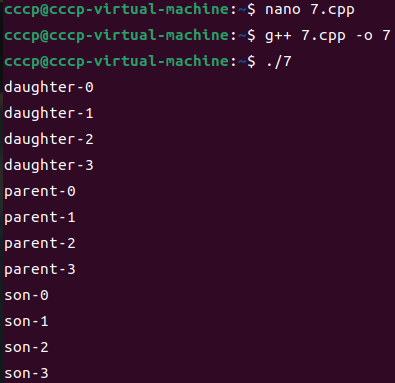
      }

   }

    return 0;

}

**运行结果:**

****

**思考题:**

**（1）进程加锁和未上锁的输出结果相同吗？ 为什么？**

**lockf()函数允许将文件区域用作信号量（监视锁），或用于控制对锁定进程的访问（强制模式记录锁定）。试图访问已锁定资源的其他进程将返回错误或进入休眠状态，直到资源解除锁定为止。当关闭文件时，将释放进程的所有锁定，即使进程仍然有打开的文件。当进程终止时，将释放进程保留的所有锁定。**

**综上所述，当lockf()上锁后处理器只运行当前进程，直至解锁为止才会运行其他进程。若不上锁，多个进程并发运行，争抢处理器资源，程序将有随机性，导致结果不同。**

**未上锁程序如下:**

**该程序删除了上锁函数，并让循环次数增加，让每一个进程尽可能地消耗时间片，可以更直观地反映出没有lockf()下的程序有随机性。**

#include <stdio.h>

#include <unistd.h>

#include<iostream>

using namespace std;

int main()

{

   int pid1, pid2;

   pid1 = fork();

   if(pid1==0)

   {

      for(int i = 0; i < 100; i++)printf("daughter-%d\n", i);

   }

   else

   {

      pid2 = fork();

      if(pid2==0)

      {

        for(int i = 0; i < 100; i++)printf("son-%d\n", i);

      }

      else

      {

        for(int i = 0; i < 100; i++)printf("parent-%d\n", i);

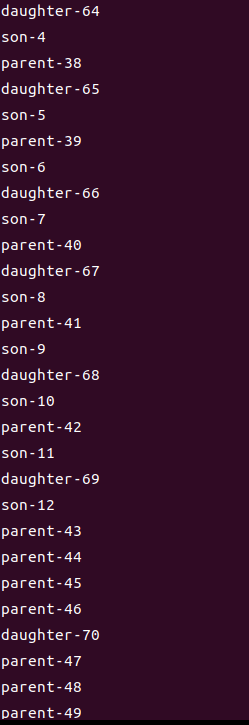
      }

   }

    return 0;

}

**未上锁程序运行结果的部分截取，可见有进程交替运行:**

****

**4、编写程序：用fork( )创建两个子进程，再用系统调用signal( )让父进程捕捉键盘上来的中断信号（即按^c键）；捕捉到中断信号后，父进程用系统调用kill( )向两个子进程发出信号，子进程捕捉到信号后分别输出下列信息后终止：**

**实验代码:**

#include<stdio.h>

#include<stdlib.h>

#include<signal.h>

int k1;

void int\_fun1()

{

    k1=0;

}

main()

{

    int k,p1;

    while((p1=fork())==-1);//创建一个子进程

    if(p1>0)//父进程

    {

        for(k=1;k<4;k++)

        {

            printf("How are you!\n");

            sleep(1);//延时函数sleep() ,延时1秒

        }

        kill(p1,12);//向子进程发送一个信号

        wait(0);

        printf("OK!\n");

        exit(0);

    }

    Else//子进程

    {

//子进程接收到父进程的信号12后执行int\_fun1,使子进程退出循环

        signal(12,(\_\_sighandler\_t)int\_fun1);

        k1=1;

        while(k1==1)

        {

            printf("I am child\n");

            sleep(1);

        }

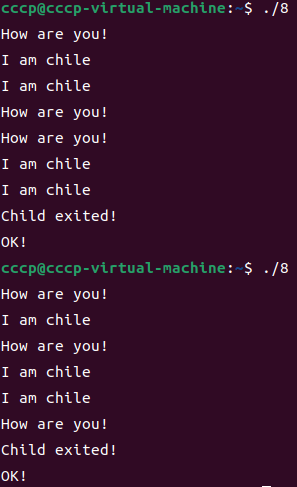
        printf("Child exited!\n");

        exit(0);

    }

}

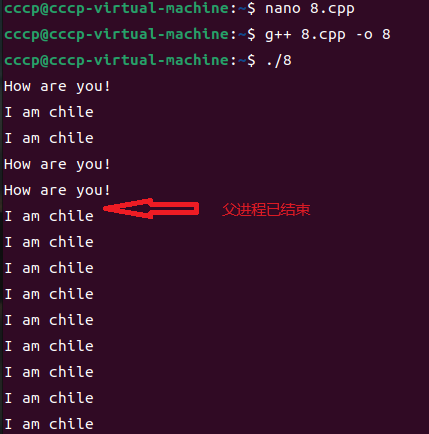
**运行结果:**

****

**思考题：**

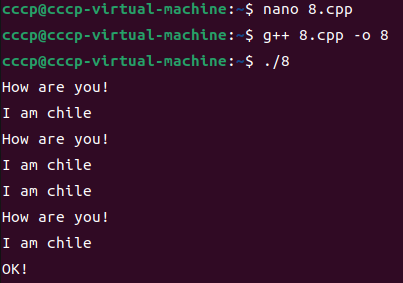
**1、如果把kill()函数注释掉，结果将如何？为什么？**

**注释掉kill()后，父进程无法向子进程发送信号，因此子进程在死循环中，每隔1秒打印”I am Child”，即使父进程结束，子进程仍会继续运行。**

****

**2、分别注释掉代码中的信号函数会有怎样的结果，为什么？**

**注释掉信号函数，子进程将无法接收父进程发送过来的信号，因此子进程会进入死循环，当父进程执行kill时终结子进程后，子进程结束运行。**

****

**3、把第一个sleep(1)改成sleep(10)又会有什么样的结果，为什么？**

**父进程的打印的时间间隔变长，sleep()是引入时延的函数，当参数变大时，时延边长。**

**4、如果一定要子进程先执行，如何改写？**

**（1）可以让创建子进程fork()执行完后，父进程使用sleep()进入一个较长的时延。**

**（2）可以创建子进程后，父进程调用wait()进入睡眠状态，子进程先执行，子进程后续唤醒父进程。**

**5、使用系统调用msgget( ),msgsnd( ),msgrev( ),及msgctl( )编制一长度为１k的消息发送和接收的程序，并分析消息的创建、发送和接收机制及控制原理。**

**数据发送代码:**

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include<stdio.h>

#include<iostream>

#define MSGKEY 75

struct  msgform

{   long  mtype;

     char  mtext[1000];

}msg;

int  msgqid;

void client()

{

    int i;

    msgqid=msgget(MSGKEY,0777);   /\*打开75#消息队列\*/

    for(i=10;i>=1;i--)

    {

        msg.mtype=i;

        printf(“(client)sent\n”);

        msgsnd(msgqid,&msg,1024,0);     /\*发送消息\*/

    }

    exit(0);

}

int main()

{

  client();

return 0;

}

**数据接收代码:**

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include<stdio.h>

#include<iostream>

#define MSGKEY 75

struct  msgform

{

    long  mtype;

    char  mtext[1000];

}msg;

int  msgqid;

void server( )

{

    msgqid=msgget(MSGKEY,0777|IPC\_CREAT);  /\*创建75#消息队列\*/

    do

    {

        msgrcv(msgqid,&msg,1030,0,0);   /\*接收消息\*/

        printf(“(server)received\n”);

    }while(msg.mtype!=1);

    msgctl(msgqid,IPC\_RMID,0);  /\*删除消息队列，归还资源\*/

    exit(0);

}

int main()

{

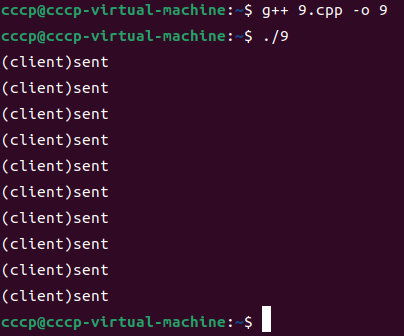
    server();

    return 0;

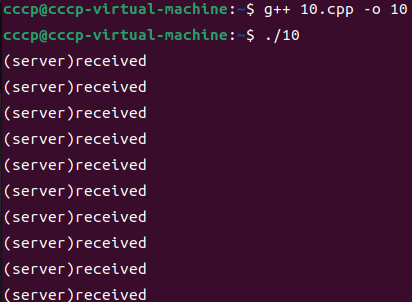
}

**运行结果:**

**发送:**

****

**接收:**

****

**思考题：**

**（1）为了便于操作和观察结果，需要编制几个程序分别用于消息的发送与接收？**

**需要编制两个程序分别用于消息的发送和接收。一个用于消息的发送，一个用于消息的接收。**

**（2）这些程序如何进行编辑、编译和执行？为什么？**

**发送程序:先建立发送队列，再向发送队列中发送消息**

**接收程序:先建立接收队列，再从队列中接收消息**

**若先执行发送程序再执行接收程序，消息发出后才建立接收队列，消息无法被接收程序接收。**

**若先执行接收程序再执行发送程序，消息发送前，接收队列以创建，接收程序能够接收程序。**

**（3）如何实现消息的发送与接收的同步？**

**要实现消息的同步，可以使用信号量和互斥锁来保护共享资源，确保发送和接收操作的互斥执行。**

**6、编制一长度为1k的共享存储区发送和接收的程序，并设计对该共享存储区进行互斥访问及进程同步的措施，必须保证实现正确的通信。**

#include<sys/types.h>

#include<sys/shm.h>

#include<sys/ipc.h>

#include<sys/wait.h>

#include<stdio.h>

#include<unistd.h>

#include<iostream>

#include<stdlib.h>

#define shmkey 75//共享存储区名字

int shmid;

int i;

int \*addr;

void client()

{

    int i;

    shmid = shmget(shmkey,1024,0777);//创建共享存储区

    int\* addr = reinterpret\_cast<int\*>(shmat(shmid, 0, 0));//共享存储区所附接到的进程虚地址

    for(i = 9;i>=0;i--)

    {

        while(\*addr!=-1);//确保服务端接收到了一条信息

        printf("(client)sent\n");

        \*addr = i;

    }

    exit(0);

}

void server()

{

    shmid = shmget(shmkey,1024,0777|IPC\_CREAT);//创建共享存储区

    int\* addr = reinterpret\_cast<int\*>(shmat(shmid, 0, 0)); //共享存储区所附接到的进程虚地址

    do

    {

        \*addr = -1;

        while(\*addr==-1);/\*响应客户端，addr到0退出循环\*/

        printf("(server)received\n");

    }while(\*addr);

    shmctl(shmid,IPC\_RMID,0); /\*撤消共享存储区，归还资源\*/

    exit(0);

}

int main()

{

    while((i=fork())==-1);

    if(!i)server();

    system("ipcs -m");

    while((i = fork())==-1);

    if(!i)client();

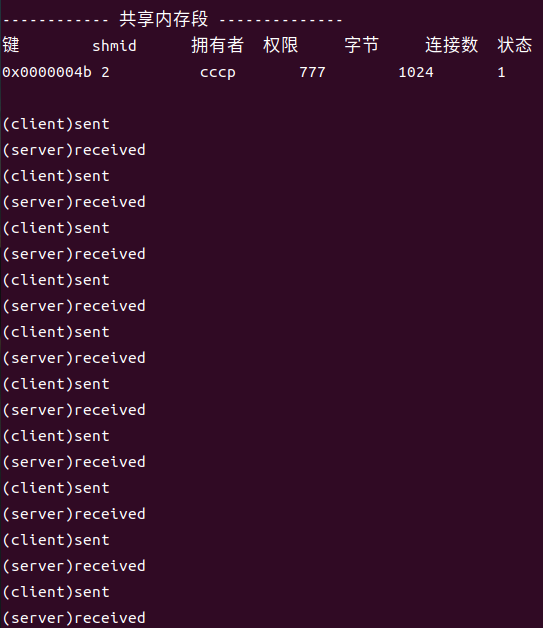
    wait(0);

    wait(0);

    return 0;

}

**运行结果:**

****

**思考题：**

**（1）为了便于操作和观察结果，需要如何合理设计程序来实现子进程间的共享存储区通信？**

**建立了两个进程server和client，系统开辟了一个共享资源区，进程都可以 进行读写操作，通过使用信号，信号量，互斥锁等等来进行同步，双方都处 于不断同步的状态，client等待资源区被server修改，而server修改后，也在 等待client修改资源区，client察觉资源区被修改，也会继续修改，两者都会 互相观察，进行多次循环。**

**（2）比较消息通信和共享存储区通信这两种进程通信机制的性能和优缺点。**

**①消息通信性能低于共享存储区通信，优点是信息的通信传递能力很强，缺点是系统需要设置多个缓冲区，占用内存空间，系统需要大量处理缓冲区的调度和管理。**

**②共享存储区通信的优点是速度快，函数接口简单，数据的共享使进程间的 数据不用传送，直接访问内存，加快了程序的效率，缺点就是共享内存没有提供同步的机制，使得使用共享内存的时候需要借助其他的方法才能进行进程间的同步工作**

**四、实验思考**

**该实验探索了进程管理与进程之间如何实现通信的问题，其中包括如何使用fork()创建子进程、wait()使进程进入睡眠、通过msgget(),msgsnd(),msgrev(),及msgctl()实现进程通信和进程同步的问题。**