实验题目：进程管理及进程通信

姓名：王波　　学号：19122557　　实验日期：2021 年 9 月 28 日

进程管理及进程通信

【实验目的】

1. 利用 Linux 提供的系统调用设计程序，加深对进程概念的理解。
2. 体会系统进程调度的方法和效果。
3. 了解进程之间的通信方式以及各种通信方式的使用。

【实验环境】

虚拟机中 Linux 环境（CentOS7 镜像）

【实验方法】

用 vi 编写 c 程序(假定程序文件名为 prog1.c),编译程序 ：$ gcc -o prog1.o prog1.c 或 $ cc -o prog1.o prog1.c 运行 ： $./prog1.o

【实验步骤、结果截图】

1. 编写程序。显示进程的有关标识（进程标识、组标识、用户标识等）。经过 5 秒钟后，执行另一个程序，最后按用户指示（如：Y/N）结束操作。

```
#include<stdlib.h>
#include<unistd.h>
main()
{
        printf("process id=%d\n",getpid());
        printf("process group id=%d\n",getpgrp());
        printf("calling process's real user id=%d\n",getuid());
        sleep(5);
        int child_pid;
        child_pid=fork();
        if(child_pid==0)
        {
          execlp("echo","echo","hello world\n",(char*)0);
          perror("exec1 error.\n");
          exit(1);
        }
        int i;
        i=wait(0);
        printf("if you want to stop this process?\n");
        char x;
        scanf("%c",&x);
        if(x=='y')
          exit(0);

}
```

```
process id=5532
process group id=5532
calling process's real user id=0
hello world

if you want to stop this process?
y
```

2. 参考例程 1，编写程序。实现父进程创建一个子进程。体会子进程与父进程分别获得不同返回值，进而执行不同的程序段的方法。

master@master:/home/master

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int i;
        if (fork()) {
          i=wait();
          printf("It is parent process.\n");
          printf("The child process,ID number %d, is finished.\n",i);}
        else{
          printf("It is child process.\n");
          sleep(10);
          exit();}
         return 0;
}
```

思考：子进程是如何产生的？ 又是如何结束的？子进程被创建后它的运行环境是怎样建立的？

答：子进程是由父进程用 fork()函数创建形成的，通过 exit（）函数自我结束，子进程被创建的后核心将其分配一个进程表项和进程标识符，检查同时运行的进程数目，并且拷贝进程项目表项的数据，由子进程继承父进程的所有文件。

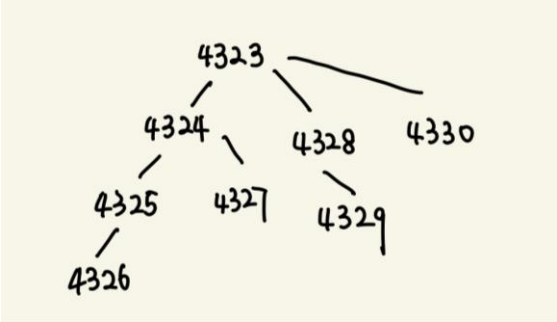
3. 参考例程 2，编写程序。父进程通过循环语句创建若干子进程。探讨进程的家族树以及子进程继承父进程的资源的关系。

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        int i,j;
        printf("My pid is %d, my father's pid is %d\n",getpid(),getppid());
        for(i=0;i<3;i++)
          if(fork()==0)
            printf("%d pid=%d ppid=%d\n", i,getpid(),getppid());
          else
          { j=wait(0);
            printf("%d:The chile %d is finished.\n" ,getpid(),j);
          }
        return 0;
}
```

```
My pid is 4323, my father's pid is 3448
0 pid=4324 ppid=4323
1 pid=4325 ppid=4324
2 pid=4326 ppid=4325
4325: The child 4326 is finished.
4324: The child 4325 is finished.
2 pid=4327 ppid=4324
4324: The child 4327 is finished.
4323: The child 4324 is finished.
1 pid=4328 ppid=4323
2 pid=4329 ppid=4328
4328: The child 4329 is finished.
4323: The child 4328 is finished.
2 pid=4330 ppid=4323
4323: The child 4330 is finished.
```

思考：① 画出进程的家族树。子进程的运行环境是怎样建立的？反复运行此程序看会有什么情况？解释一下。



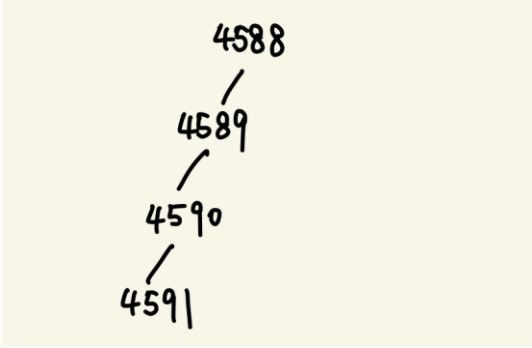
子进程的运行环境是由将其创建的父进程而建立的，反复运行程序会发现每个进程标识号在不断改变，这是因为同一时间有许多进程在被创建。

② 修改程序，使运行结果呈单分支结构，即每个父进程只产生一个子进程。画出进程树，解释该程序。

```c
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
int main()
{
        int i;
        printf("My pid is %d,my father's pid is %d\n",getpid(),getppid());
        for(i=0;i<3;i++)
        {
          if(fork())
          {
            wait();
            exit(0);
          }
          else
          {
            printf("%d pid=%d ppid=%d\n",i,getpid(),getppid());
          }

        }
        return 0;
}
```

```
[root@master master]# vim 3.3.c
[root@master master]# vim 3.3.1.c
[root@master master]# gcc -o 3.3.1.o 3.3.1.c
[root@master master]# ./3.3.1.o
My pid is 4588,my father's pid is 3448
0 pid=4589 ppid=4588
1 pid=4590 ppid=4589
2 pid=4591 ppid=4590
```



解释：当该进程为父进程时就创建子进程并退出，当该进程为子进程时返回标识号。

4. 参考例程 3 编程，使用 fork()和 exec()等系统调用创建三个子进程。子进程分别启动不同程序，并结束。反复执行该程序，观察运行结果，结束的先后，看是否有不同次序。

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
        int child_pid1,child_pid2,child_pid3;
        int pid,status;
        setbuf(stdout,NULL);
        child_pid1=fork();
        if(child_pid1==0)
        { execlp("echo","echo","child process 1",(char *)0);
          perror("exec1 error.\n ");
          exit(1);
        }
        child_pid2=fork();
        if(child_pid2==0)
        { execlp("date","date",(char *)0);
          perror("exec2 error.\n ");
          exit(2);
        }
        child_pid3=fork();
        if(child_pid3==0)
        { execlp("ls","ls",(char *)0);
          perror("exec3 error.\n ");
          exit(3);
```

```
        puts("Parent process is waiting for chile process return!");
        while((pid=wait(&status))!=-1)
        { if(child_pid1==pid) /*若子进程1 结束*/
          printf("child process 1 terminated with status %d\n",(status>>8));
          else
            {if(child_pid2==pid) /*若子进程2 结束*/
                printf("child process 2 terminated with status %d\n",(status>>8));
             else
                { if(child_pid3==pid) /*若子进程3 结束*/
                     printf("child process 3 terminated with status %d\n" ,(status>>8));
                }
            }
        }
        puts("All child processes terminated.");
        puts("Parent process terminated.");
        exit(0);
        return 0;
}
```

```
[root@master master]# ./3.4.o
Parent process is waiting for chile process return!
2021年 09月 30日 星期四 11:23:45 CST
child process 2 terminated with status 0
]        3-2.c    3.3.c  4-1.c    f         mytext2.dat  openssl-1.0.2f         Pictures  图片   桌面
2.c      3-2.cpp  3.3.o  4.c      f1        mytext3.dat  openssl-1.0.2f.tar.gz  公共      文档
3.2.1.c  3.3.1.c  3.4.c  countf1  mima.txt  mytext.txt   passwd                 模板      下载
3.2.1.o  3.3.1.o  3.4.o  Desktop  mytext    OpenMP       passwd.c               视频      音乐
child process 3 terminated with status 0
child process 1
child process 1 terminated with status 0
All child processes terminated.
Parent process terminated.
```

思考：子进程运行其它程序后，进程运行环境怎样变化的？反复运行此程序看会有什么情况？解释一下。

答：子进程运行其他程序后，这个进程就完全被新程序代替。由于并没有产生新进程所以进程标识号不改变，除此之外的旧进程的其他信息，代码段，数据段，栈段等均被新程序的信息所代替。

新程序从自己的 main()函数开始运行。反复运行此程序发现结束的先后顺序是不可预知的，每次运行的结果不一样。原因是当每个子进程运行其他程序时，他们的结束随着其他程序的结束而结束，所以结束的先后次序在改变。

5. 参考例程 4 编程，验证子进程继承父进程的程序、数据等资源。如用父、子进程修改公共变量和私有变量的处理结果；父、子进程的程序区和数据区的位置。

```c
#include<sys/types.h>
#include<unistd.h>
int globa=4;
int main()
{
        pid_t pid;
        int vari=5;
        printf("before fork.\n");
        if ((pid=fork())<0)
        {
          printf("fork error.\n");
          exit(0);
        }

        else if(pid==0)
        { /*子进程执行*/
          globa++;
          vari--;
          printf("Child %d changed the vari and globa.\n",getpid());
        }
        else /*父进程执行*/
          printf("Parent %d did not changed the vari and globa.\n",getpid());
        printf("pid=%d, globa=%d, vari=%d\n",getpid(),globa,vari); /*都执行*/
        exit(0);
}
"3.5.c" 26L, 516C
```

```
[root@master master]# ./3.5.o
before fork.
Parent 5744 did not changed the vari and globa.
pid=5744, globa=4, vari=5
Child 5745 changed the vari and globa.
pid=5745, globa=5, vari=4
```

思考：子进程被创建后，对父进程的运行环境有影响吗？解释一下。

答：子进程被创建后，对父进程的运行环境无影响，因为子进程在运行时，他有自己的代码和数据段，这些都可以作修改，但是父进程的代码段和数据段是不会随着子进程的数据段和代码段的改变而改变。

6. 参照《实验指导》第五部分中“管道操作的系统调用”。复习管道通信概念，参考例程 5，编写一个程序。父进程创建两个子进程，父子进程之间利用管道进行通信。要求能显示父进程、子进程各自的信息，体现通信效果。

```c
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
main()
{
        int i,r,j,k,l,p1,p2,fd[2];
        char buf[50],s[50];
        pipe(fd);
        while((p1=fork())==-1);
        if(p1==0)
         {
          lockf(fd[1],1,0);
          sprintf(buf,"Child process P1 is sending messages! \n");
          printf("Child process P1! \n");
          write(fd[1],buf,50);
          lockf(fd[1],0,0);
          sleep(5);
          j=getpid();
          k=getppid();
          printf("P1 %d is weakup. My parent process ID is %d.\n",j,k);
          exit(0);
        }
        else
        { while((p2=fork())==-1);
          if(p2==0)
```

```c
        else
        { while((p2=fork())==-1);
          if(p2==0)
          {
            lockf(fd[1],1,0);
            sprintf(buf,"Child process P2 is sending messages! \n");
            printf("Child process P2! \n");
            write(fd[1],buf,50);
            lockf(fd[1],0,0);
            sleep(5);
            j=getpid();
            k=getppid();
            printf("P2 %d is weakup. My parent process ID is %d.\n",j,k);
            exit(0);
          }
          else
          { l=getpid();
            wait(0);
            if(r=read(fd[0],s,50)==-1)
              printf("Can't read pipe. \n");
            else
              printf("Parent %d: %s \n",l,s);
            wait(0);
```

```c
          else
          { l=getpid();
            wait(0);
            if(r=read(fd[0],s,50)==-1)
              printf("Can't read pipe. \n");
            else
              printf("Parent %d: %s \n",l,s);
            wait(0);
            if(r=read(fd[0],s,50)==-1)
              printf("Can't read pipe. \n");
            else
              printf("Parent %d: %s \n",l,s);
            exit(0);
          }
        }
}
```

```
[root@master master]# ./3.6.o
Child process P1!
Child process P2!
P1 6030 is weakup. My parent process ID is 6029.
Parent 6029: Child process P1 is sending messages!

P2 6031 is weakup. My parent process ID is 6029.
Parent 6029: Child process P2 is sending messages!
```

思考：①什么是管道？进程如何利用它进行通信的？解释一下实现方法。

答：管道是指能够连接一个写进程和一个读进程，并允许他们以生产者—消费者方式进行通信的一个共享文件，又称 pipe 文件。由写进程从管道的入端将数据写入管道，而读进程则从管道出端读出数据来进行通信。

②修改睡眠时机、睡眠长度，看看会有什么变化。请解释。

答：修改睡眠时间和睡眠长度都会引起进程被唤醒的时间不一，因为睡眠时机决定进程在何时睡眠，睡眠长度决定进程何时被唤醒。

③加锁、解锁起什么作用？不用它行吗？

答：加锁、解锁是为了解决临界资源的共享问题。不用它将会引起无法有效管理数据，即数据会被修改导致读错了数据。

7. 编程验证：实现父子进程通过管道进行通信。进一步编程，验证子进程结束，由父进程执行撤消进程的操作。测试父进程先于子进程结束时，系统如何处理“孤儿进程”的。

将上一程序中的父进程中的 wait()函数去掉即可。

```
[root@master master]# gcc -o 3.7.o 3.7.c
[root@master master]# ./3.7.o
Child process P1!
Parent 6591: Child process P1 is sending messages!

Child process P2!
Parent 6591: Child process P2 is sending messages!
```

思考：对此作何感想，自己动手试一试？解释一下你的实现方法。

只要在父进程后加上 wait()函数，然后打印“子进程已经结束”，一旦子进程结束，父进程撤销进程。当父进程先于子进程终止时，所有子进程的父进程改变为 init 进程称为由 init 进程领养。

8. 编写两个程序一个是服务者程序，一个是客户程序。执行两个进程之间通过消息机制通信。消息标识 MSGKEY 可用常量定义，以便双方都可以利用。客户将自己的进程标识（pid）通过消息机制发送给服务者进程。服务者进程收到消息后，将自己的进程号和父进程号发送给客户，然后返回。客户收到后显示服务者的 pid 和 ppid，结束。以下例程 6 基本实现以上功能。这部分内容涉及《实验指导》第五部分中“IPC 系统调用”。先熟悉一下，再调试程序。

```c
/*The server receives the message from client,and answer a message*/
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
struct msgform
{
  long mtype;
  char mtext[256];
}msg;
int msgqid;
main()
{
  int i,pid,* pint;
  extern cleanup();
  for(i=0;i<20;i++)
    signal(i,cleanup);
  msgqid=msgget(MSGKEY,0777|IPC_CREAT);
  for(;;)
  {
    msgrcv(msgqid,&msg,256,1,0);
    pint=(int *)msg.mtext;
    pid=*pint;
    printf("Server :receive from pid %d\n",pid);
    msg.mtype=pid;

    printf("Server :receive from pid %d\n",pid);
    msg.mtype=pid;
    *pint=getpid();
    msgsnd(msgqid,&msg,sizeof(int),0);
  }
}
cleanup()
{
  msgctl(msgqid,IPC_RMID,0);
  exit(0);
}
```

```c
/*The client send a message to server,and receives another message from
 * server*/
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 75
struct msgform
{
  long mtype;
  char mtext[256];
};
main()
{
  struct msgform msg;
  int msgqid,pid,*pint;
  msgqid=msgget(MSGKEY,0777);
  pid=getpid();
  pint=(int *)msg.mtext;
  *pint=pid;
  msg.mtype=1;
  msgsnd(msgqid,&msg,sizeof(int),0);
  msgrcv(msgqid,&msg,256,pid,0);
  printf("Clint :  receive from pid %d\n",* pint);
}
```

```
[root@master master]# ./server.o
Server :receive from pid 7084
Server :receive from pid 7091
Server :receive from pid 7098
```

```
[root@master master]# ./client.o
Clint :  receive from pid 6909
[root@master master]# ./client.o
Clint :  receive from pid 6909
[root@master master]# ./client.o
Clint :  receive from pid 6909
```

思考：想一下服务者程序和客户程序的通信还有什么方法可以实现？解释一下你的设想，有兴趣试一试吗。

答：还可以用信号量机制来实现。信号量是一个整形计数器，用来控制多个进程对共享资源的访问。或者通过消息队列信号机制，通过向消息队列发送信息、接收信息来实现进程间的通信。

9. 这部分内容涉及《实验指导》第五部分中“有关信号处理的系统调用”。编程实现软中断信号通信。父进程设定软中断信号处理程序，向子进程发软中断信号。子进程收到信号后执行相应处理程序。

```
#include<unistd.h>
#include<sys/types.h>
main()
{
  int i,j,k;
  int func();
  signal(18,func());
  if(i=fork())
  {
    j=kill(i,18);
    printf("Parent: signal 18 has been sent to child %d,returned %d.\n",i,j);
    k=wait(); /*父进程被唤醒*/
    printf("After wait %d,Parent %d: finished.\n",k,getpid());
  }
  else
  {
    sleep(10);
    printf("Child %d: A signal from my parent is recived.\n",getpid());
  } /*子进程结束，向父进程发子进程结束信号*/
}
func() /*处理程序*/
{ int m;
  m=getpid();
  printf("I am Process %d: It is signal 18 processing function.\n",m);
}
-- 插入 --
```

```
[root@master master]# ./3.9.o
I am Process 7627: It is signal 18 processing function.
Parent: signal 18 has been sent to child 7628,returned 0.
After wait 7628,Parent 7627: finished.
[root@master master]#
```

思考：这就是软中断信号处理，有点儿明白了吧？讨论一下它与硬中断有什么区别？看来还挺管用，好好利用它。

答：硬中断是由外部硬件产生的，而软中断是 CPU 根据软件的某条指令或者软件对标志寄存器的某个标志位的设路而产生的。

10.怎么样，试一下吗？用信号量机制编写一个解决生产者—消费者问题的程序，这可是受益匪浅的事。本《实验指导》第五部分有关进程通信的系统调用中介绍了信号量机制的使用。

```c
#define N 5
#define M 10

int in=0;
int out=0;
int buff[M]={0};

sem_t empty_sem;
sem_t full_sem;
pthread_mutex_t mutex;

int product_id=0;
int prochase_id=0;

void Handlesignal(int signo){
        printf("程序退出\n",signo);
        exit(0);
}
```

```c
void print(){
        int i;
        printf("产品队列为");
        for(i=0;i<M;i++)
                printf("%d",buff[i]);
        printf("\n");
}

void *product(){
        int id=++product_id;
        while(1){
                sleep(2);
                sem_wait(&empty_sem);
                pthread_mutex_lock(&mutex);
                in=in%M;
                printf("生产者%d在产品队列中放入第%d个产品\t",id,in);
                buff[in]=1;
                print();
                ++in;
                pthread_mutex_unlock(&mutex);
                sem_post(&full_sem);
        }
}
```

```c
void *prochase(){
        int id=++prochase_id;
        while(1){
                sleep(5);
                sem_wait(&full_sem);
                pthread_mutex_lock(&mutex);
                out=out%M;
                printf("消费者%d从产品队列中取出第%d个产品\t",id,out);
                buff[out]=0;
                print();
                ++out;
                pthread_mutex_unlock(&mutex);
                sem_post(&empty_sem);
        }
}

int main(){
        printf("生产者和消费者数目都为5，产品缓冲为10，生产者每2秒生产一个产品，消费者每5秒消费一个产品，Ctrl+退出程序\n");
        pthread_t id1[N];
        pthread_t id2[N];
        int i;
        int ret[N];
```

```
        if(signal(SIGNT,Handlesignal)==SIG_ERR){
                printf("信号安装出错");
        }
        int ini1=sem_init(&empty_sem,0,M);
        int ini2=sem_init(&full_sem,0,0);
        if(ini1&&ini2!=0){
                printf("信号量初始化失败！\n");
                exit(1);
        }
        int ini3=pthread_mutex_init(&mutex,NULL);
        if(ini3!=0){
                printf("线程同步初始化失败！\n");
                exit(1);
        }
        for(i=0;i<N;i++){
                ret[i]=pthread_create(&id1[i],NULL,product,(void*)(&i));
                if(ret[i]!=0){
                        printf("生产者%d线程创建失败!\n",i);
                        exit(1);
                }
        }
```

```
        for(i=0;i<N;i++){
                ret[i]=pthread_create(&id2[i],NULL,prochase,NULL);
                if(ret[i]!=0){
                        printf("消费者%d线程创建失败！\n",i);
                        exit(1);
                }
        }
        for(i=0;i<N;i++){
                pthread_join(id1[i],NULL);
                pthread_join(id1[i],NULL);
        }
        exit(0);
}
```

```
[root@master master]# gcc -pthread 10.c -o 10.out
[root@master master]# ./10.o
bash: ./10.o: 没有那个文件或目录
[root@master master]# ./10.out
生产者和消费者数目都为5，产品缓冲为10，生产者每2秒生产一个产品，消费者每5秒消费一个产品，Ctrl+退出程序
生产者1在产品队列中放入第0个产品          产品队列为1000000000
生产者2在产品队列中放入第1个产品          产品队列为1100000000
生产者3在产品队列中放入第2个产品          产品队列为1110000000
生产者4在产品队列中放入第3个产品          产品队列为1111000000
生产者5在产品队列中放入第4个产品          产品队列为1111100000
生产者1在产品队列中放入第5个产品          产品队列为1111110000
生产者2在产品队列中放入第6个产品          产品队列为1111111000
生产者3在产品队列中放入第7个产品          产品队列为1111111100
生产者4在产品队列中放入第8个产品          产品队列为1111111110
生产者5在产品队列中放入第9个产品          产品队列为1111111111
消费者1从产品队列中取出第0个产品          产品队列为0111111111
消费者2从产品队列中取出第1个产品          产品队列为0011111111
消费者3从产品队列中取出第2个产品          产品队列为0001111111
消费者4从产品队列中取出第3个产品          产品队列为0000111111
消费者5从产品队列中取出第4个产品          产品队列为0000011111
生产者1在产品队列中放入第0个产品          产品队列为1000011111
生产者2在产品队列中放入第1个产品          产品队列为1100011111
```

【实验心得】

本次实验是有关进程管理和进程通信的程序设计，在编写程序的过程中，我掌握了 Linux 提供的系统调用，加深了对进程的理解，明白了父进程和子进程的调用关系，通过 fork 可以创建子进程，exit 可以结束进程，wait 可以等待其余进程的结束，exec 使子进程执行其他程序；了解了子进程对变量的修改并不影响父进程，原因是子进程继承父进程的程序、数据等资源，有自己的代码和数据段，对父进程无影响；也知道了父进程在子进程之前结束时，子进程由 init 进程领养。

通过本次实验，我还加深了对管道、消息队列、共享内存通信、信号量等概念的理解，通过编写消费者—生产者程序，让我对临界资源、上锁、解锁、进程同步有了更深的认识。

【研究及讨论】

1. 讨论 Linux 系统进程运行的机制和特点，系统通过什么来管理进程？

在 Linux 中，每个进程在创建时都会被分配一个数据结构，称为进程控制块（Process Control Block，简称 PCB）。PCB 中包含了很多重要的信息，供系统调度和进程本身执行使用。所有进程的 PCB 都存放在内核空间中。PCB 中最重要的信息就是进程 PID，内核通过这个 PID 来唯一标识一个进程。PID 可以循环使用，最大值是 32768。init 进程的 pid 为 1，其他进程都是 init 进程的后代。

除了进程控制块（PCB）以外，每个进程都有独立的内核堆栈（8k），一个进程描述符结构，这些数据都作为进程的控制信息储存在内核空间中；而进程的用户空间主要存储代码和数据。 Linux 操作系统使用一些系统调用（如：fork()、wait、exit 等）来实现对进程的管理。

2. C 语言中是如何使用 Linux 提供的功能的？用程序及运行结果举例说明。

C 语言是通过在.c 文件中添加函数调用的.h 文件，来调用进程管理的相关函数，并通过 getpid()和 getppid（）来查看 Linux 系统为每个进程分配的进程号。

```
#include<stdio.h>
#include<unistd.h>
main(){
        int i,j;
        printf("My pid is %d,my father's pid is %d\n",getpid(),getppid());
        for(i=0;i<3;i++){
                if(fork()==0)
                        printf("%d pid=%d ppid=%d\n",i,getpid(),getppid());
                else
                {
                        j=wait(0);
                        printf("%d:The child %d is finished.\n",getpid(),j);
                }
        }
}
```

```
[root@master master]# vim d-2.c
[root@master master]# gcc -o d-2.o d-2.c
[root@master master]# ./d-2.o
My pid is 3176,my father's pid is 3096
0 pid=3177 ppid=3176
1 pid=3178 ppid=3177
2 pid=3179 ppid=3178
3178:The child 3179 is finished.
3177:The child 3178 is finished.
2 pid=3180 ppid=3177
3177:The child 3180 is finished.
3176:The child 3177 is finished.
1 pid=3181 ppid=3176
2 pid=3182 ppid=3181
3181:The child 3182 is finished.
3176:The child 3181 is finished.
2 pid=3183 ppid=3176
3176:The child 3183 is finished.
[root@master master]#
```

3. 什么是进程？如何产生的？举例说明。

进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

引起进程创建的事件有用户登录、作业调度、提供服务和应用请求。一旦操作系统发现了要求创建新进程的事件后，便调用进程创建原语 Creat( )按下述步骤创建一个新进程。申请空白 PCB、为新进程分配资源、初始化进程控制块、将新进程插入就绪队列

例如用户登入，在分时系统中，用户在终端键入登录命令后，如果是合法用户，系统将为该终端建立一个进程，并把它插入就绪队列中。

4. 进程控制如何实现的？举例说明。

进程控制一般是由 OS 的内核中的原语来实现的。

例子：唤醒原语 wakeup( ) 当被阻塞进程所期待的事件出现时，如 I/O 完成或其所期待的数据已经到达，则由有关进程(比如用完并释放了该 I/O 设备的进程)调用唤醒原语 wakeup( )，将等待该事件的进程唤醒。

唤醒原语执行的过程是：首先把被阻塞的进程从等待该事件的阻塞队列中移出，将其 PCB 中的现行状态由阻塞改为就绪，然后再将该 PCB 插入到就绪队列中。

5. 进程通信方式各有什么特点？用程序及运行结果举例说明。

进程间通信可分为 4 种形式：

(1) 主从式：① 主进程可自由地使用从进程的资源或数据；② 从进程的动作受主进程的控制；③ 主进程和从进程的关系是固定的。

(2) 会话式：① 使用进程在使用服务进程所提供的服务之前，必须得到服务进程的许可；② 服务进程根据使用进程的要求提供服务，但对所提供服务的控制由服务进程自身完成。③ 使用进程和服务进程在通信时有固定连接关系。

(3) 消息或邮箱机制：① 只要存在空缓冲区或邮箱，发送进程就可以发送消息。② 与会话系统不同，发送进程和接收进程之间无直接连接关系，接收进程可能在收到某个发送进程发来的消息之后，又转去接收另一个发送进程发来的消息。③ 发送进程和接收进程之间存在缓冲区或邮箱用来存放被传送消息。

(4) 共享存储区方式：①诸进程可通过对共享存储区中数据的读或写来实现通信。②进程在通信前，先向系统申请获得共享存储区中的一个分区，并指定该分区的关键字；若系统已经给其他进程分配了这样的分区，则将该分区的描述符返回给申请者。③由申请者把获得的共享存储分区连接到本进程上。④可像读、写普通存储器一样地读、写该公用存储分区。

例子：父进程创建两个子进程，父子进程之间利用管道进行通信。

```c
#include<stdio.h>
#include<stdlib.h>
int main(){
        int i,r,j,k,l,p1,p2,fd[2];
        char buf[50],s[50];
        pipe(fd);
        while((p1=fork())==-1);
        if(p1==0)
        {
                lockf(fd[1],1,0);
                sprintf(buf,"Child process P1 is sending message!\n");
                printf("Child process P1!\n");
                write(fd[1],buf,50);
                lockf(fd[1],0,0);
                sleep(5);
                j=getpid();
                k=getppid();
                printf("P1 %d is weakup.My parent process ID is %d.\n",j,k);
                exit(0);
        }
        else{
                while((p2=fork())==-1);
                if(p2==0){
```

2,18 顶

```c
        else{
                while((p2=fork())==-1);
                if(p2==0){
                        lockf(fd[1],1,0);
                        sprintf(buf,"Child proces P2 is sending message!\n");
                        printf("Child process P2!\n");
                        write(fd[1],buf,50);
                        lockf(fd[1],0,0);
                        sleep(5);
                        j=getpid();
                        k=getppid();
                        printf("P2 %d is weakup.My parent process ID is %d\n",j,k);
                        exit(0);
                }
                else{
                        l=getpid();
                        wait(0);
                        if(r=read(fd[0],s,50)==-1)
                                printf("Can't read pipe.\n");
                        else
                                printf("Parent %d:%s\n",l,s);
                        wait(0);
                        if(r=read(fd[0],s,50)==-1)
                                printf("Can't read pipe.\n");
                        else

                }
                else{
                        l=getpid();
                        wait(0);
                        if(r=read(fd[0],s,50)==-1)
                                printf("Can't read pipe.\n");
                        else
                                printf("Parent %d:%s\n",l,s);
                        wait(0);
                        if(r=read(fd[0],s,50)==-1)
                                printf("Can't read pipe.\n");
                        else
                                printf("Parent %d :%s \n",l,s);
                        exit(0);
                }
        }
```

```
[root@master master]# vim d-3.c
[root@master master]# gcc -o d-3.o d-3.c
[root@master master]# ./d-3.o
Child process P2!
Child process P1!
P2 3310 is weakup.My parent process ID is 3308
Parent 3308:Child proces P2 is sending message!

P1 3309 is weakup.My parent process ID is 3308.
Parent 3308 :Child process P1 is sending message!
```

6. 管道通信如何实现？该通信方式可以用在何处？

向管道(共享文件)提供输入的发送进程(即写进程)，以字符流形式将大量的数据送入管道；而接受管道输出的接收进程(即读进程)，则从管道中接收(读)数据。 这种通信方式用于数据传输、资源共享和事件通知。

7. 什么是软中断？软中断信号通信如何实现？

软中断是利用硬件中断的概念，用软件方式进行模拟，实现宏观上的异步执行效果。 利用 signal 和 kill 实现软中断通信。

kill(pid,signal)：向进程 pid 发送信号 signal，若 pid 进程在可中断的优先级（低优先级）上睡眠，则将其唤醒。

signal(sig,ps)：设置 sig 号软中断信号的处理方式； SIG_DFL：系统默认方式，一般是终止进程； SIG_IGN：忽略（屏蔽）；func()： 用制定义函数 func()处理。 Signal 设珞的处理方式，仅一次有效，处理后即回到默认方式。