进程控制

实验一：

要求：

通过系统调用 fork()函数，编写一个 C 程序，以便在子进程中实现 Collatz 猜想：任意写出一个正整数 N，并按照以下规律变化直到变换为 1：如果是奇数，下一步变成 3N+1；如果是偶数，下一步变成 N/2。

需求分析：

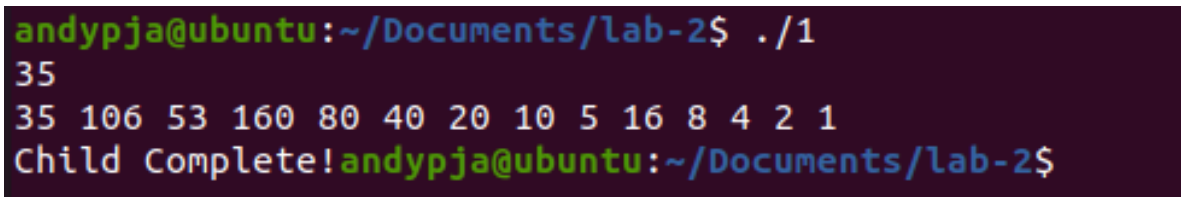
通过命令行获得用户输入的启动数字，之后创建一个子进程，在子进程中根据输入的数字生成 Collatz 数列，父进程等待子进程结束之后再退出。父进程和子进程之间不存在信息传递。

程序源代码：

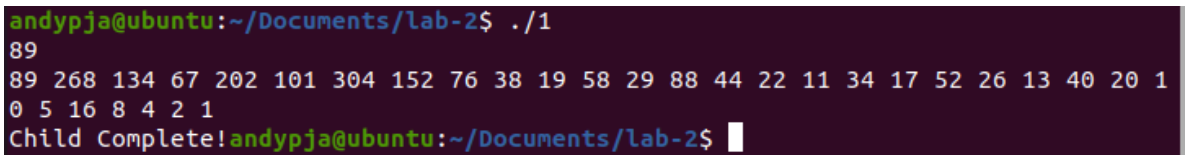
```
#include<stdio.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
    int num;
    scanf("%d",&num);
    pid_t pid;
    int status;
    pid=fork();
    if(pid==0)
    {
            printf("%d ",num);
        while(num!=1)
        {
            if(num%2==0)
            {
                num=num/2;
            }
            else
            {
                num=num*3+1;
            }
            printf("%d ",num);
        }
        printf("\n");
        exit(0);
    }
```

```
    else
    {
        wait(&status);
        printf("Child Complete!");
        exit(0);
    }
}
```

程序测试：
测试一：



测试二：



实验二：
要求：

通过内存共享技术实现 Collatz 数列的产生

需求分析：

因为要使用内存共享技术，因此再父进程和子进程之间创建一个共享内存对象，在子进程中生成所求的序列，然后在子进程中将这个序列写入内存共享对象。之后父进程等待子进程结束之后，从内存共享对象中读取相应的序列，然后将这个序列输入到屏幕上。最终需要删除这片共享内存区域。

程序源代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>
#include<sys/wait.h>

struct shareMemory{
```

```
    int col[200];
};

int main(int argc, const char *argv[])
{
    key_t    key_n ;
    int      shm_id;
    pid_t    pid;
    int status;
    void *shm = NULL;   //共享内存的首地址
    struct shareMemory *shared;
    int num;
    scanf("%d",&num);

    if((key_n = ftok("/",'s')<0))   //生成共享内存的 key 值
    {
        perror("Fail to ftok");
        exit(EXIT_SUCCESS);
    }

    if((shm_id = shmget(key_n,1024,0666|IPC_CREAT))==-1)   //创建共享内存
    {
        perror("Fail to shmget");
        exit(EXIT_SUCCESS);
    }

    if((pid = fork())==-1)
    {
        perror("Fail to fork");
        exit(EXIT_SUCCESS);
    }

    if(pid==0)                                              //子进程，用来读
    {
        int i=1;
        shm = shmat(shm_id,0,0);                            //将共享内存连接到当前进程的
地址空间
        if(shm==(void *)-1)
        {
            perror("Fail to shmat");
            exit(EXIT_SUCCESS);
        }

        shared = (struct shareMemory *)shm;             //设置共享内存
```

```
        shared->col[1]=num;
        while(num!=1)
        {
            if(num%2==0)
            {
                num=num/2;
        }
        else
        {
            num=num*3+1;
        }
        i++;
        shared->col[i]=num;
}
shared->col[0]=i;
        if(shmdt(shm)==-1)    //把共享内存从当前进程分离
        {
            perror("Fail to shmdt");
            exit(EXIT_SUCCESS);
        }
        exit(0);
}

if(pid!=0)
{
        wait(&status);
        shm = shmat(shm_id,0,0);
        if(shm==(void *)-1)
        {
            perror("Fail to shmat");
            exit(EXIT_SUCCESS);
        }
        shared = (struct shareMemory *)shm;
        int j=1;
        for(j=1;j<=shared->col[0];j++)
        {
            printf("%d ",shared->col[j]);
        }



        if(shmdt(shm)==-1)
        {
            perror("Fail to shmdt");
```

```
                exit(EXIT_SUCCESS);
            }
            exit(EXIT_SUCCESS);
        }

        return 0;
}
```

程序测试：
测试一：

```
andypja@ubuntu:~/Documents/lab-2$ ./2
35
35 106 53 160 80 40 20 10 5 16 8 4 2 1 andypja@ubuntu:~/Documents/lab-2$
```

测试二：

```
andypja@ubuntu:~/Documents/lab-2$ ./2
89
89 268 134 67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 1
0 5 16 8 4 2 1 andypja@ubuntu:~/Documents/lab-2$
```

实验三：
要求：

设计一个程序，通过普通管道进行通信，要求一个进程发送一个字符串消息给第二个进程，第二个进程收到消息之后，变更字符串中的字母大小写，然后再发送回第一个进程。

需求分析：

在程序中需要创建两个管道用于通信，其中第一个管道父进程使用写端，子进程使用读端，用于父进程向子进程发送原始的字符串；第二个管道父进程使用读端，子进程使用写端，用于在子进程将原始字符串进行相应的修改之后通过这个管道将修改之后的字符串发送给父进程，最终父进程将修改后的字符串和原始字符串输出到屏幕中。

程序源代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
#include<sys/wait.h>

int main()
{
    int ret1,ret2;
    int fd1[2];
    int fd2[2];
```

```
    int status;
    pid_t pid;
    ret1=pipe(fd1);
    ret2=pipe(fd2);
    pid=fork();
    if(pid>0)
    {
        char msg[50]="My name is Pja,my English name is Andy";
        char result[50];
        close(fd1[0]);
        write(fd1[1],msg,strlen(msg)+1);
        wait(&status);
        close(fd2[1]);
        read(fd2[0],result,50);
        printf("The orign string is: %s\n",msg);
        printf("The string after change is:%s",result);
        close(fd1[1]);
        close(fd2[2]);
    }
    else
    {
        char read_msg[50];
        close(fd1[1]);
        read(fd1[0],read_msg,50);
        close(fd2[0]);
        int i;
        for(i=0;i<strlen(read_msg);i++)
        {
            if(read_msg[i]>='a'&&read_msg[i]<='z')
            {
                read_msg[i]=read_msg[i]-32;
            }
            else if(read_msg[i]>='A'&&read_msg[i]<='Z')
            {
                read_msg[i]=read_msg[i]+32;
            }
        }
        write(fd2[1],read_msg,strlen(read_msg)+1);
        close(fd2[1]);
        close(fd1[0]);
        exit(0);
    }
}
```

程序测试：

测试结果：

```
andypja@ubuntu:~/Documents/lab-2$ ./3
The orign string is: My name is Pja,my English name is Andy
The string after change is:mY NAME IS pJA,MY eNGLISH NAME IS aNDYandypja@ubuntu
:~/Documents/lab-2$
```