实验报告

实验环境

Linux系统

实验题目

题目一

1. 源代码

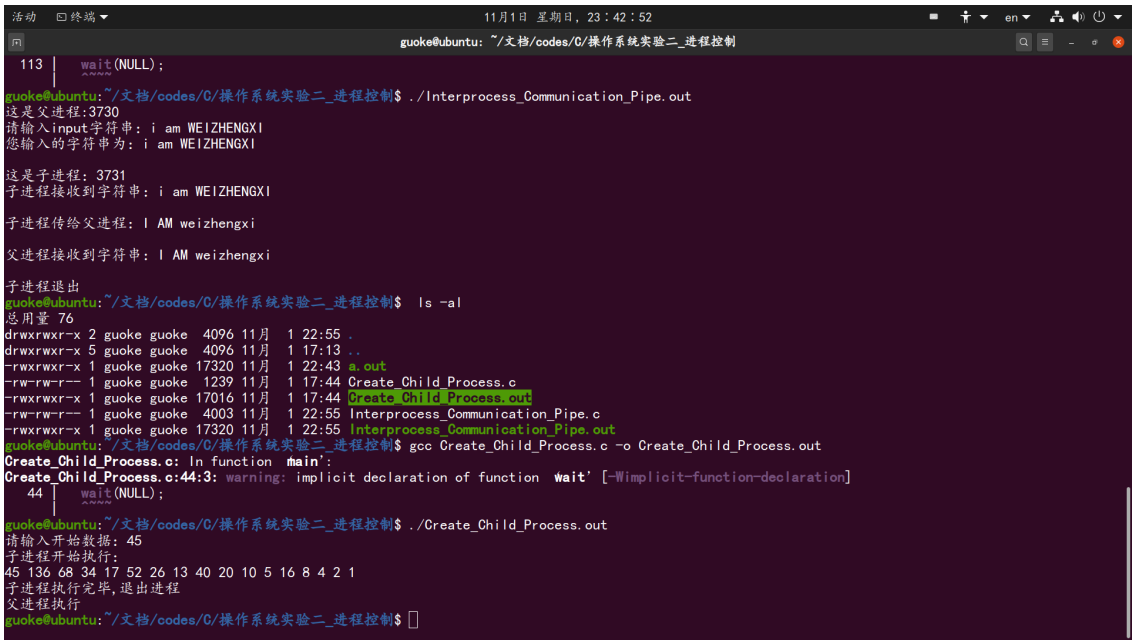
```
#include<sys/types.h>
 #include<unistd.h>
 #include<stdio.h>
 #include<stdlib.h>

 int Collatz(int num);

 int main(void)
 {
     int ret = 0;

     pid_t pid;      //进程id
     int num = 0;

     printf("请输入开始数据: ");
     scanf("%d",&num);  //读入开始数据

     /* 创建子进程 */
     pid = fork();

     /* 进程创建出错 */
     if(pid < 0)
     {
         ret = -1;
         printf("func fork() err:%d\n",ret);
         return ret;
     }
     /* 子进程执行程序 */
     else if(pid == 0)
     {
         printf("子进程开始执行: \n");
         ret = Collatz(num);         //根据起始数据输出Collatz猜想数列
             if(ret != 0)
             {
                     printf("func Collatz() err:%d\n",ret);
                     return ret;
             }
         printf("子进程执行完毕,退出进程\n");
         exit(0);            //退出子进程
     }
     /* 父进程执行程序 */
```

```
    else
    {
        wait(NULL);
        printf("父进程执行\n");
    }

    return ret;
}

int Collatz(int num)
{
    int ret = 0;
    int tmp = num;

    /* 判断输入是不是正整数 */
    if(num <= 0)
    {
        ret = -1;
        printf("func Collatz() err:%d\n",ret);
        return ret;
    }

    /* Collatz猜想 */
    while(tmp != 1)
    {
        printf("%d ",tmp);

        if(tmp%2 == 0)
        {
            tmp = tmp/2;
        }
        else
        {
            tmp = 3 * tmp + 1;
        }

    }
    printf("%d\n",tmp);

    return ret;
}
```

2. 分析

    1. `int Collatz(int num)`
        - 参数： 起始数字
        - 返回值： 错误码。正确返回0，错误返回-1
    2. `fork()`
        - 子进程中返回值为0
        - 父进程中返回子进程的进程id
        - 返回小于0表示出错

3. 结果



题目二

1. 源代码
2. 分析
3. 结果

题目三

1. 源代码

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

//缓冲区长度
#define BUFFER_LENGTH 256

int ToggleCase(char *input,int in_length,char *output,int* out_length);

int main(void)
{
    int ret = 0;
    int pipefd1[2] = {0};
    int pipefd2[2] = {0};
    pid_t pid = 0;
    char buf[BUFFER_LENGTH] = {0};
    char tmp_buf[BUFFER_LENGTH] = {0};
    int tmp_buf_length = 0;

    /* 创建两个匿名管道 */
    if(pipe(pipefd1) < 0 )
    {
        ret = -1;
        printf("func pipe() err:%d(pipe(pipefd1) < 0)\n",ret);
        return ret;
```

```
    }
    if(pipe(pipefd2) < 0 )
        {
                ret = -1;
                printf("func pipe() err:%d(pipe(pipefd2) < 0)\n",ret);
                return ret;
        }

    /* 创建子进程 */
    pid = fork();
    //创建子进程失败
    if(pid  < 0)
    {
        ret = -2;
        printf("func fork() err:%d(pid = fork() < 0)\n",ret);
        return ret;
    }
    //子进程执行程序
    else if(pid == 0)
    {
        close(pipefd1[0]);  //关闭管道1的读文件功能
        close(pipefd2[1]);  //关闭管道2的写文件功能

        //读通道2的文件
        if(read(pipefd2[0],buf,BUFFER_LENGTH) < 0)
        {
            ret = -1;
            printf("func read() err:%d(read(pipefd[1],buf,BUFFER_LENGTH) <
0)\n",ret);
            return ret;
        }
        printf("这是子进程：%d\n",getpid());
        printf("子进程接收到字符串：%s\n",buf);

        //将字符串大小写转换
        ret = ToggleCase(buf,strlen(buf),tmp_buf,&tmp_buf_length);
        if(ret != 0)
            {
                    ret = -1;
                    printf("func ToggleCase() err:%d\n",ret);
                    return ret;
            }

        //将转换后的字符串传给父进程
        printf("子进程传给父进程：%s\n",tmp_buf);
                if(write(pipefd1[1],tmp_buf,strlen(tmp_buf)) < 0)
                {
                        ret = -1;
                        printf("func write()
err:%d(write(pipefd1[1],tmp_buf,strlen(tmp_buf))\n",ret);
                        return ret;
                }

        //退出子进程
        exit(0);


    }
```

```
    //父进程执行程序
    else
    {
        printf("这是父进程:%d\n",getpid());
        close(pipefd1[1]);  //关闭管道1的写文件功能
        close(pipefd2[0]);  //关闭管道2的读文件功能

        //命令行读入字符串
        printf("请输入input字符串：");
            fgets(buf,BUFFER_LENGTH,stdin);
            printf("您输入的字符串为：%s\n",buf);

        //写入通道2
        if(write(pipefd2[1],buf,strlen(buf)) < 0)
        {
            ret = -1;
            printf("func write() err:%d(write(pipefd2[1],buf,strlen(buf)) <
0)\n",ret);
            return ret;
        }

        //等待通道1有进程写入信息
        if(read(pipefd1[0],tmp_buf,BUFFER_LENGTH) < 0)
                {
                        ret = -1;
                        printf("func read()
err:%d(read(pipefd1[1],tmp_buf,BUFFER_LENGTH) < 0)\n",ret);
                        return ret;
                }
                printf("父进程接收到字符串：%s\n",tmp_buf);
        //等待子进程退出
        wait(NULL);
        printf("子进程退出\n");

    }

    return ret;
}

int ToggleCase(char *input,int in_length, char* output, int* out_length)
{
    int ret = 0;
    int  i = 0;

    //参数检查
    if(output == NULL || in_length < 0)
    {
        ret = -1;
        printf("func ToggleCase() err:%d(output == NULL || in_length <
0)\n",ret);
        return ret;
    }

    //大小写转化
    for(i = 0; i < in_length; i++)
    {
        if(input[i] <= 'z' && input[i] >= 'a')
        {
```

```
            output[i] = input[i] - (int)('a'-'A');
        }
        else if(input[i] <= 'Z' && input[i] >= 'A')
        {
            output[i] = input[i] + (int)('a' - 'A');
        }
        else
        {
            output[i] = input[i];
        }
    }
    output[i] = '\0';

    *out_length = in_length;

    return ret;

}
```

2. 分析
    1. `int ToggleCase(char *input,int in_length, char* output, int* out_length)`
        - 参数input：传入的字符串
        - 参数input_length：传入的字符串的长度
        - 参数output：将传入的字符串大小写反转之后存储与output
        - 参数output_length：传出的字符串的长度
        - 返回值：错误码。正确返回0，错误返回-1；
    2. `pipe(int* pipefd)`
        - 参数：整形二维数组，执行函数后保存文件描述符，pipefd[0]为读文件，pipefd[1]为写文件
    3. `read()`
        - 读通道（文件），产生阻塞。
    4. `write()`
        - 写通道（文件），不产生阻塞。
3. 结果