|  |
| --- |
| BUPT |
| 操作系统实验二 |
| 进程控制 |

|  |
| --- |
| 姓名：陈朴炎 学号：2021211138  2023-10-22 |

目录

[实验二 进程控制 2](#_Toc149070374)

[1 实验内容 2](#_Toc149070375)

[1.1 实验一内容 2](#_Toc149070376)

[1.2 实验二内容 2](#_Toc149070377)

[1.3 实验三内容 2](#_Toc149070378)

[2 环境配置 2](#_Toc149070379)

[2.1 使用WSL在Windows下安装Linux 2](#_Toc149070380)

[2.2 Ubuntu下的vscode下载 3](#_Toc149070381)

[3 实验一 3](#_Toc149070382)

[3.1 实验内容 3](#_Toc149070383)

[3.2 主要系统调用 3](#_Toc149070384)

[3.3 程序设计 4](#_Toc149070385)

[3.4 程序源代码 4](#_Toc149070386)

[3.5 测试报告/运行结果 5](#_Toc149070387)

[3.6 结果分析 10](#_Toc149070388)

[4 实验二 11](#_Toc149070389)

[4.1 实验二内容 11](#_Toc149070390)

[4.2 主要系统调用 11](#_Toc149070391)

[4.3 程序设计 13](#_Toc149070392)

[4.4 程序源代码 13](#_Toc149070393)

[4.5 测试用例及结果 16](#_Toc149070394)

[4.6 结果分析 38](#_Toc149070395)

[5 实验三 39](#_Toc149070396)

[5.1 实验三内容 39](#_Toc149070397)

[5.2 主要系统调用 39](#_Toc149070398)

[5.3 程序设计 40](#_Toc149070399)

[5.4 程序源代码 41](#_Toc149070400)

[5.5 测试用例及结果 43](#_Toc149070401)

[5.6 结果分析 44](#_Toc149070402)

# 实验二 进程控制

## 1 实验内容

Collatz猜想：任意写出一个正整数N，并且按照以下的规律进行变换：如果是个奇数，则下一步变成3N+1；如果是个偶数，则下一步变成N/2。无论N是怎样的一个数字，最终都无法逃脱回到谷底1。例如：如果N=35，则有序列35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1。

### 1.1 实验一内容

采用系统调用fork()，编写一个C程序，以便在子进程中生成这个序列。要求如下：

（1）从命令行提供启动数字

（2）由子进程输出数字序列

（3）父进程等子进程结束后再退出。

### 1.2 实验二内容

以共享内存技术编程实现Collatz猜想。要求在父子进程之间建立一个共享内存对象，允许子进程将序列内容写入共享内存对象，当子进程完成时，父进程输出序列。父进程包括如下步骤：建立共享内存对象（shm\_open(), ftruncate(), mmap()）建立子进程并等待他终止输出共享内存的内容删除共享内存对象。

### 1.3 实验三内容

普通管道通信设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一个进程。比如，第一个进程发消息：“I am Here”，第二个进程收到后，将它改变为：“i AM hERE”之后，再发给第一个进程。提示：

（1）需要创建子进程，父子进程之间通过普通管道进行通信。

（2）需要建立两个普通管道。

## 2 环境配置

本次实验在Windows下通过WSL来使用Linux环境。

gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0

### 2.1 使用WSL在Windows下安装Linux

先决条件：

必须运行 Windows 10 版本 2004 及更高版本（内部版本 19041 及更高版本）或 Windows 11 才能使用以下命令。

在管理员模式下打开 PowerShell 或 Windows 命令提示符，方法是右键单击并选择“以管理员身份运行”，输入 wsl --install 命令，然后重启计算机。如图2-1。此命令将启用运行 WSL 并安装 Linux 的 Ubuntu 发行版所需的功能。

|  |
| --- |
| PowerShell |
| wsl --install |

图2-1 在Windows下以管理员身份安装Linux

### 2.2 Ubuntu下的vscode下载

在终端里依次输入下列命令，提示输入a即可：

|  |
| --- |
| Ubuntu |
| sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make |
| sudo apt-get update |
| sudo apt-get install ubuntu-make |
| umake ide visual-studio-code |

图2-2 Ubuntu命令行下的vscode下载

下载好后，cd进入实验的目标文件夹位置，输入命令code . 就能用vscode打开该文件夹了。

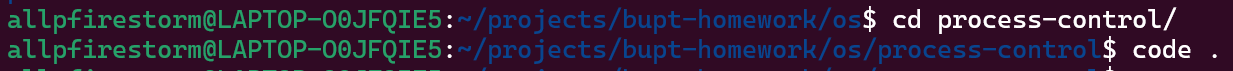


图2-3 用vscode打开文件夹

至此，实验的准备工作做完了，下面就开始真正的实验步骤。

## 3 实验一

### 3.1 实验内容

采用系统调用fork()，编写一个C程序，以便在子进程中生成这个序列。要求：（1）从命令行提供启动数字

（2）由子进程输出数字序列

（3）父进程等子进程结束后再退出。

### 3.2 主要系统调用

1、fork()函数

在调用 fork 之后，会生成一个新的进程，称为子进程，它相当于父进程的副本。子进程继承了父进程的大部分状态，包括代码、数据、打开的文件描述符、用户ID、工作目录等。子进程是在父进程的基础上复制出来的。

fork 的返回值不同，对于父进程来说，fork 返回子进程的PID（进程标识符），而对于子进程来说，fork 返回0。这是因为父进程和子进程需要区分彼此，通过返回值来判断是哪个进程在执行。

子进程和父进程是并发执行的，它们在不同的地址空间中执行，彼此不受影响。子进程可以独立地修改自己的数据，而不会影响父进程的数据。

2、waitpid()函数

waitpid(pid\_t pid, int \*status, int options)

waitpid 函数是一个用于等待子进程结束的系统调用，它允许父进程等待特定的子进程或任何子进程的结束。这是一个在多进程编程中非常有用的函数，可以用来确保父进程在子进程结束后继续执行。

参数说明：

pid：要等待的子进程的PID，可以有以下取值：

> 0：等待具有指定PID的子进程。

-1：等待任何子进程，相当于 wait 函数。

0：等待与当前进程组ID相同的任何子进程。

< -1：等待指定进程组ID的任何子进程。

status：用于获取子进程的退出状态信息。如果不需要这些信息，可以传入 NULL。子进程在exit时可以附带上退出信息，让父进程接受。

options：等待选项，通常可以设置为0。

waitpid 函数的主要作用是等待子进程的结束。它会暂停父进程的执行，直到一个子进程结束，然后返回该子进程的PID，并将子进程的退出状态信息存储在 status 中。父进程可以根据 status 中的信息来判断子进程是否正常退出、以及子进程的退出状态等。

### 3.3 程序设计

在主函数中，先让用户输入一个任意的数字。在主线程中多开一个线程。若是父进程，则等待子线程工作完毕，并在最后输出提示信息。若是子进程，则根据用户输入的数字计算出序列信息，计算过程如下：

1. 若an是1，则停止
2. 若an是偶数，则an+1 = an / 2
3. 若an是奇数，则an+1 = 3\*an + 1

同时，也得注意线程安全，如果pid < 0则表示fork函数出错，如果在父进程中WIFEXITED函数返回的是0，则代表子进程退出错误。

### 3.4 程序源代码

|  |
| --- |
| C |
| #include<stdio.h>  #include<sys/types.h>  #include<unistd.h>  #include<sys/wait.h>  int main()  {  int num=0;  printf("Please input the number: ");  scanf("%d",&num);  pid\_t pid = fork();  if(pid>0){//这个是父进程  printf("Parent Process pid: %d\n",getpid());  int status;  waitpid(pid, &status, 0);  if(WIFEXITED(status)){  printf("child process has done, and exit.\n");  }else{  fprintf(stderr, "child process exit error.\n");  }  printf("father process has exit.\n");  }else if(pid==0){//在子进程中输出序列  printf("Child Process pid: %d\n",getpid());  while(num!=1)  {  printf("child process ouput the number: ");  if(num%2==0)  {  num=num/2;  printf("%d ",num);  }  else  {  num=3\*num+1;  printf("%d ",num);  }  printf("\n");  }  }else{  perror("Fork falied.\n");  }  return 0;  } |

### 3.5 测试报告/运行结果

编译程序，并运行，如图3-1。



图3-1 Ubuntu下编译C源程序并运行

#### 测试用例1

输入56，序列如图3-2，依次是：28、14、7、22、11、34、17、52、26、16、40、20、10、5、16、8、4、2、1。子进程结束后打印出“child process has done, and exit.”，父进程等待子进程结束后打印“father process has exit.”。

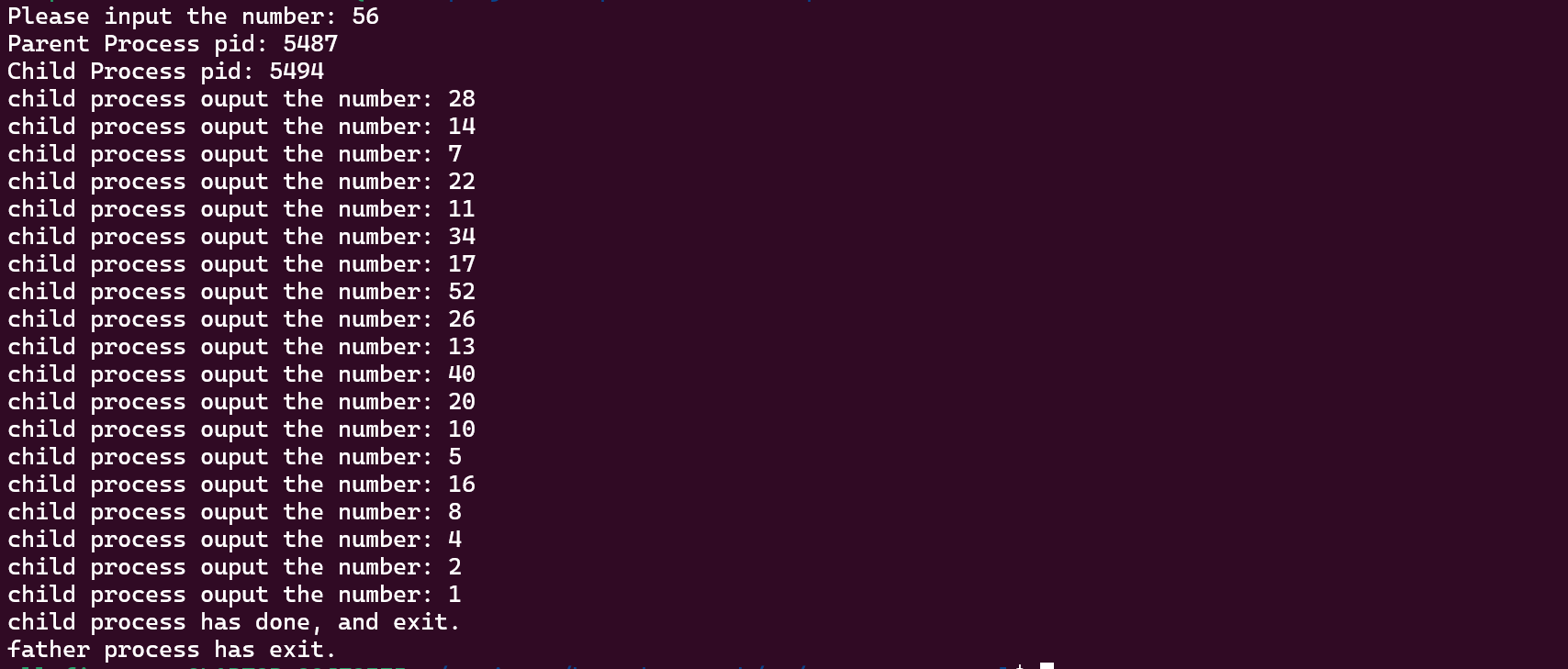


图3-2 测试结果1

#### 测试用例2

如图3-3测试结果2，输入1，程序输出父进程和子进程的id号，之后子进程接受到1后直接退出，父进程等待子进程退出后打印出”father process has exit.”

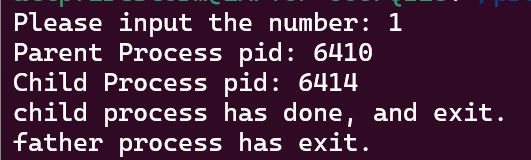


图3-3 测试结果2

#### 测试用例3

如图3-4，输入35，程序输出父进程和子进程的id号，之后子进程接收到35后输出序列：106、53、160、80、40、20、10、5、16、8、4、2、1。子进程结束后打印”child process has done, and exit.”，父进程等待子进程接收后打印出”father process has exit.”。

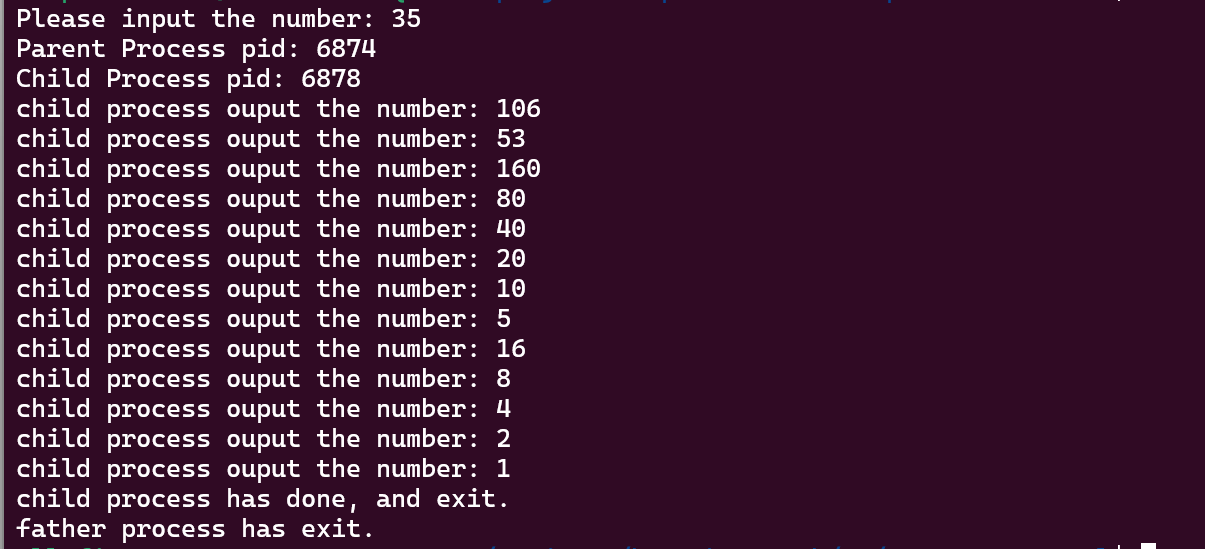


图3-4 测试结果2

#### 测试用例4

如下输出结果，输入6553，程序输出父进程和子进程的id号，之后子进程接收到6553后输出序列：19660、9830、4915、14746、7373、……、106、53、160、80、40、20、10、5、16、8、4、2、1。子进程结束后打印”child process has done, and exit.”，父进程等待子进程接收后打印出”father process has exit.”。

|  |
| --- |
| 输入6553的输出结果 |
| Please input the number: 6553  Parent Process pid: 2416  Child Process pid: 2423  child process ouput the number: 19660  child process ouput the number: 9830  child process ouput the number: 4915  child process ouput the number: 14746  child process ouput the number: 7373  child process ouput the number: 22120  child process ouput the number: 11060  child process ouput the number: 5530  child process ouput the number: 2765  child process ouput the number: 8296  child process ouput the number: 4148  child process ouput the number: 2074  child process ouput the number: 1037  child process ouput the number: 3112  child process ouput the number: 1556  child process ouput the number: 778  child process ouput the number: 389  child process ouput the number: 1168  child process ouput the number: 584  child process ouput the number: 292  child process ouput the number: 146  child process ouput the number: 73  child process ouput the number: 220  child process ouput the number: 110  child process ouput the number: 55  child process ouput the number: 166  child process ouput the number: 83  child process ouput the number: 250  child process ouput the number: 125  child process ouput the number: 376  child process ouput the number: 188  child process ouput the number: 94  child process ouput the number: 47  child process ouput the number: 142  child process ouput the number: 71  child process ouput the number: 214  child process ouput the number: 107  child process ouput the number: 322  child process ouput the number: 161  child process ouput the number: 484  child process ouput the number: 242  child process ouput the number: 121  child process ouput the number: 364  child process ouput the number: 182  child process ouput the number: 91  child process ouput the number: 274  child process ouput the number: 137  child process ouput the number: 412  child process ouput the number: 206  child process ouput the number: 103  child process ouput the number: 310  child process ouput the number: 155  child process ouput the number: 466  child process ouput the number: 233  child process ouput the number: 700  child process ouput the number: 350  child process ouput the number: 175  child process ouput the number: 526  child process ouput the number: 263  child process ouput the number: 790  child process ouput the number: 395  child process ouput the number: 1186  child process ouput the number: 593  child process ouput the number: 1780  child process ouput the number: 890  child process ouput the number: 445  child process ouput the number: 1336  child process ouput the number: 668  child process ouput the number: 334  child process ouput the number: 167  child process ouput the number: 502  child process ouput the number: 251  child process ouput the number: 754  child process ouput the number: 377  child process ouput the number: 1132  child process ouput the number: 566  child process ouput the number: 283  child process ouput the number: 850  child process ouput the number: 425  child process ouput the number: 1276  child process ouput the number: 638  child process ouput the number: 319  child process ouput the number: 958  child process ouput the number: 479  child process ouput the number: 1438  child process ouput the number: 719  child process ouput the number: 2158  child process ouput the number: 1079  child process ouput the number: 3238  child process ouput the number: 1619  child process ouput the number: 4858  child process ouput the number: 2429  child process ouput the number: 7288  child process ouput the number: 3644  child process ouput the number: 1822  child process ouput the number: 911  child process ouput the number: 2734  child process ouput the number: 1367  child process ouput the number: 4102  child process ouput the number: 2051  child process ouput the number: 6154  child process ouput the number: 3077  child process ouput the number: 9232  child process ouput the number: 4616  child process ouput the number: 2308  child process ouput the number: 1154  child process ouput the number: 577  child process ouput the number: 1732  child process ouput the number: 866  child process ouput the number: 433  child process ouput the number: 1300  child process ouput the number: 650  child process ouput the number: 325  child process ouput the number: 976  child process ouput the number: 488  child process ouput the number: 244  child process ouput the number: 122  child process ouput the number: 61  child process ouput the number: 184  child process ouput the number: 92  child process ouput the number: 46  child process ouput the number: 23  child process ouput the number: 70  child process ouput the number: 35  child process ouput the number: 106  child process ouput the number: 53  child process ouput the number: 160  child process ouput the number: 80  child process ouput the number: 40  child process ouput the number: 20  child process ouput the number: 10  child process ouput the number: 5  child process ouput the number: 16  child process ouput the number: 8  child process ouput the number: 4  child process ouput the number: 2  child process ouput the number: 1  child process has done, and exit.  father process has exit. |

### 3.6 结果分析

不管输入什么数字，最终序列都会回到1。在网上搜集资料，发现这是一个数学猜想，目前还没能证明对于任何数，都会回到1。目前，人们对于小于1e+18的数都验证了该猜想。

主进程：

主进程开始执行，并等待用户输入一个整数（num）。主进程使用 fork 函数创建了一个子进程。在父进程中，fork 返回子进程的进程ID。父进程使用 waitpid 函数等待子进程的结束。waitpid 函数允许父进程等待子进程的终止并获取子进程的退出状态。如果子进程成功退出，父进程使用 WIFEXITED 宏检查退出状态。如果子进程正常退出，父进程输出相应的信息。最后，父进程结束执行。

子进程：

在子进程中，首先打印出子进程的进程ID，以便区分父子进程。子进程使用一个循环来计算Collatz猜想的序列，直到输入的数字变为1。它在每一步都会根据奇偶性进行不同的计算，并打印出计算的结果。子进程完成计算后，它退出。

在操作系统的控制下，父子进程之间进行协同工作，实现了任务的并发执行。

## 4 实验二

### 4.1 实验二内容

### 4.2 主要系统调用

1、shm\_open(shmName, O\_CREAT | O\_RDWR, 0777)函数

shm\_open()是用于创建或打开共享内存对象的系统调用，它用于创建或打开共享内存，以便多个进程可以共享数据。

shm\_open 的参数：

shmName: 共享内存的名称，这是一个字符串，类似文件名的标识符。

O\_CREAT | O\_RDWR: 这是标志参数，O\_CREAT 表示如果共享内存不存在，则创建它，O\_RDWR 表示以可读可写的方式打开共享内存。

0777: 这是权限参数，指定了共享内存的权限，类似于文件权限。在这里，0777 表示所有用户都有读写权限，具体权限可以根据需求进行调整。关于文件权限参数，有如下扩展：

rw------- (600) 只有拥有者有读写权限。

-rw-r--r-- (644) 只有拥有者有读写权限；而属组用户和其他用户只有读权限。

-rwx------ (700) 只有拥有者有读、写、执行权限。

-rwxr-xr-x (755) 拥有者有读、写、执行权限；而属组用户和其他用户只有读、执行权限。

-rwx--x--x (711) 拥有者有读、写、执行权限；而属组用户和其他用户只有执行权限。

-rw-rw-rw- (666) 所有用户都有文件读、写权限。

-rwxrwxrwx (777) 所有用户都有读、写、执行权限。

shm\_open 的作用：

当你调用 shm\_open 函数时，它会尝试创建一个共享内存对象（如果不存在），并返回一个文件描述符，你可以使用这个文件描述符来访问共享内存。

如果共享内存对象已经存在，那么 shm\_open 将打开已存在的共享内存对象，并返回相应的文件描述符。

错误处理：

如果 shm\_open 失败，它将返回 -1，并可以使用 perror 函数或 errno 来获取详细的错误信息，以便进行适当的错误处理。

2、ftruncate(shmFd, BUFSIZE)函数

ftruncate 是一个系统调用，用于更改文件的大小，它通常与共享内存对象一起使用，以便指定共享内存的大小。下面是关于 ftruncate 函数的详细解释：

ftruncate 的参数：

shmFd：这是共享内存对象的文件描述符，它是通过 shm\_open 打开或创建共享内存对象后获得的。

BUFSIZE：这是要将共享内存对象的大小更改为的目标大小。通常，它是一个以字节为单位的整数值。

ftruncate 的作用：

当你调用 ftruncate 函数时，它会将共享内存对象的大小更改为指定的大小（BUFSIZE）。如果指定的大小比共享内存对象的当前大小大，那么共享内存将增大，多余的部分将被初始化为零。如果指定的大小比共享内存对象的当前大小小，那么共享内存将被截断，截断后的部分数据将被删除。

错误处理：

如果 ftruncate 调用失败，它会返回 -1，并可以使用 perror 函数或 errno 获取详细的错误信息，以进行适当的错误处理。

3、(int \*)mmap(NULL, BUFSIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shmFd, 0);

mmap 是一个系统调用，用于将文件或共享内存对象映射到进程的地址空间。在你的上下文中，它用于将共享内存对象映射到进程的地址空间，以便你可以通过指针访问共享内存数据。以下是有关 mmap 函数的详细解释：

mmap 的参数：

NULL：这是期望内核自动选择映射的地址。你可以传递 NULL 以让内核选择适当的地址。

BUFSIZE：这是要映射的共享内存对象的大小，以字节为单位。

PROT\_READ | PROT\_WRITE：这是保护标志，指定了内存映射的访问权限。 PROT\_READ 表示可读权限，PROT\_WRITE 表示可写权限。

MAP\_SHARED：这个标志表示将共享内存对象映射为多个进程可以共享的内存，而不是私有内存。

shmFd：这是共享内存对象的文件描述符，是通过 shm\_open 打开或创建共享内存对象后获得的。

0：这是偏移量，通常设置为 0，表示从共享内存对象的开头开始映射。

mmap 的作用：

mmap 将共享内存对象映射到进程的地址空间，并返回一个指向映射区域的指针。一旦映射完成，你可以通过指针访问共享内存中的数据，就像访问普通内存一样。如果多个进程都映射了相同的共享内存对象，它们可以通过共享内存进行通信。

错误处理：

如果 mmap 调用失败，它会返回 MAP\_FAILED（通常是 (void\*)-1），并且你可以使用 perror 函数或 errno 获取详细的错误信息，以进行适当的错误处理。

### 4.3 程序设计

在主函数中，使用shm\_open函数创建一个共享内存对象，如果已经存在则获取它的文件描述符。这个共享内存对象用于存储Collatz序列。如果出现错误，程序会打印错误信息并退出。使用mmap函数将共享内存映射到当前进程的地址空间，得到一个指向共享内存的指针，即sharedSeq。这个指针允许程序访问共享内存中的数据。开辟了一块共享内存，名称为collatz\_shared\_memory。

要求用户输入一个整数，作为Collatz序列的初始值

创建子进程，通过fork()函数，创建子进程，在子进程中，程序计算Collatz序列，并将序列存储在共享内存中。子进程通过逐步计算Collatz序列，将每个值存储在sharedSeq数组中，直到值为1，然后退出子进程，并将子进程退出状态设置为序列的长度。

在父进程中使用waitpid等待子进程的退出，并检查子进程是否正常退出。如果子进程正常退出，父进程获取子进程的退出状态，即Collatz序列的长度，然后打印这个长度以及序列的内容。

### 4.4 程序源代码

|  |
| --- |
| c |
| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/types.h> #include <sys/mman.h> #include <sys/wait.h> #include <fcntl.h> #include <string.h> #include <errno.h>  #define BUFSIZE 2048 // 共享内存大小  int main() {     printf("=============lab2 shared memory=============\n");      printf("please input the number: ");     int start;      int seqLen = 0;     scanf("%d", &start);      // 创建共享内存对象     const char \*shmName = "collatz\_shared\_memory";     int shmFd = shm\_open(shmName, O\_CREAT | O\_RDWR, 0777);     if (shmFd == -1)      {         perror("shm\_open");          return 1;     }     if (ftruncate(shmFd, BUFSIZE) == -1)     {         perror("ftruncate error.\n");         return 1;     }     // 将共享内存映射到地址空间     int \*sharedSeq = (int \*)mmap(NULL, BUFSIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shmFd, 0);     if (sharedSeq == MAP\_FAILED)      {         perror("mmap error.\n");         return 1;     }      pid\_t childPid = fork();     if (childPid < 0)      {         perror("child process doesn't forked.\n");          return 1;     }     else if (childPid == 0)      {         printf("child process start.\n");          int index = 0;         while (start != 1 && index < BUFSIZE)         {             printf("child: %d\n",start);             sharedSeq[index++] = start;             if (start % 2 == 0)             {                 start = start / 2;             }             else             {                 start = 3 \* start + 1;             }         }         sharedSeq[index] = 1;         printf("child process has writen data to shared memory.\n");          // 子进程退出状态设置为序列长度          exit(index + 1);     }      else     {         int status;         waitpid(childPid, &status, 0);         if (WIFEXITED(status))         {             // 父进程接收子进程的序列退出长度             seqLen = WEXITSTATUS(status);             printf("child process has done, and exit.\n");              printf("seqlength in father process: %d\n", seqLen);             for (int i = 0; i < seqLen; i++)             {                 printf("father process output the seq: %d\n", sharedSeq[i]);             }         }         else         {             fprintf(stderr, "child process exit error.\n");         }     } } |

### 4.5 测试用例及结果

#### 测试用例1

|  |
| --- |
| 输入：27 |
| =============lab2 shared memory=============  please input the number: 27  child process start.  child: 27  child: 82  child: 41  child: 124  child: 62  child: 31  child: 94  child: 47  child: 142  child: 71  child: 214  child: 107  child: 322  child: 161  child: 484  child: 242  child: 121  child: 364  child: 182  child: 91  child: 274  child: 137  child: 412  child: 206  child: 103  child: 310  child: 155  child: 466  child: 233  child: 700  child: 350  child: 175  child: 526  child: 263  child: 790  child: 395  child: 1186  child: 593  child: 1780  child: 890  child: 445  child: 1336  child: 668  child: 334  child: 167  child: 502  child: 251  child: 754  child: 377  child: 1132  child: 566  child: 283  child: 850  child: 425  child: 1276  child: 638  child: 319  child: 958  child: 479  child: 1438  child: 719  child: 2158  child: 1079  child: 3238  child: 1619  child: 4858  child: 2429  child: 7288  child: 3644  child: 1822  child: 911  child: 2734  child: 1367  child: 4102  child: 2051  child: 6154  child: 3077  child: 9232  child: 4616  child: 2308  child: 1154  child: 577  child: 1732  child: 866  child: 433  child: 1300  child: 650  child: 325  child: 976  child: 488  child: 244  child: 122  child: 61  child: 184  child: 92  child: 46  child: 23  child: 70  child: 35  child: 106  child: 53  child: 160  child: 80  child: 40  child: 20  child: 10  child: 5  child: 16  child: 8  child: 4  child: 2  child process has writen data to shared memory.  child process has done, and exit.  seqlength in father process: 112  father process output the seq: 27  father process output the seq: 82  father process output the seq: 41  father process output the seq: 124  father process output the seq: 62  father process output the seq: 31  father process output the seq: 94  father process output the seq: 47  father process output the seq: 142  father process output the seq: 71  father process output the seq: 214  father process output the seq: 107  father process output the seq: 322  father process output the seq: 161  father process output the seq: 484  father process output the seq: 242  father process output the seq: 121  father process output the seq: 364  father process output the seq: 182  father process output the seq: 91  father process output the seq: 274  father process output the seq: 137  father process output the seq: 412  father process output the seq: 206  father process output the seq: 103  father process output the seq: 310  father process output the seq: 155  father process output the seq: 466  father process output the seq: 233  father process output the seq: 700  father process output the seq: 350  father process output the seq: 175  father process output the seq: 526  father process output the seq: 263  father process output the seq: 790  father process output the seq: 395  father process output the seq: 1186  father process output the seq: 593  father process output the seq: 1780  father process output the seq: 890  father process output the seq: 445  father process output the seq: 1336  father process output the seq: 668  father process output the seq: 334  father process output the seq: 167  father process output the seq: 502  father process output the seq: 251  father process output the seq: 754  father process output the seq: 377  father process output the seq: 1132  father process output the seq: 566  father process output the seq: 283  father process output the seq: 850  father process output the seq: 425  father process output the seq: 1276  father process output the seq: 638  father process output the seq: 319  father process output the seq: 958  father process output the seq: 479  father process output the seq: 1438  father process output the seq: 719  father process output the seq: 2158  father process output the seq: 1079  father process output the seq: 3238  father process output the seq: 1619  father process output the seq: 4858  father process output the seq: 2429  father process output the seq: 7288  father process output the seq: 3644  father process output the seq: 1822  father process output the seq: 911  father process output the seq: 2734  father process output the seq: 1367  father process output the seq: 4102  father process output the seq: 2051  father process output the seq: 6154  father process output the seq: 3077  father process output the seq: 9232  father process output the seq: 4616  father process output the seq: 2308  father process output the seq: 1154  father process output the seq: 577  father process output the seq: 1732  father process output the seq: 866  father process output the seq: 433  father process output the seq: 1300  father process output the seq: 650  father process output the seq: 325  father process output the seq: 976  father process output the seq: 488  father process output the seq: 244  father process output the seq: 122  father process output the seq: 61  father process output the seq: 184  father process output the seq: 92  father process output the seq: 46  father process output the seq: 23  father process output the seq: 70  father process output the seq: 35  father process output the seq: 106  father process output the seq: 53  father process output the seq: 160  father process output the seq: 80  father process output the seq: 40  father process output the seq: 20  father process output the seq: 10  father process output the seq: 5  father process output the seq: 16  father process output the seq: 8  father process output the seq: 4  father process output the seq: 2  father process output the seq: 1 |

#### 测试用例2

|  |
| --- |
| 输入56 |
| =============lab2 shared memory=============  please input the number: 56  child process start.  child: 56  child: 28  child: 14  child: 7  child: 22  child: 11  child: 34  child: 17  child: 52  child: 26  child: 13  child: 40  child: 20  child: 10  child: 5  child: 16  child: 8  child: 4  child: 2  child process has writen data to shared memory.  child process has done, and exit.  seqlength in father process: 20  father process output the seq: 56  father process output the seq: 28  father process output the seq: 14  father process output the seq: 7  father process output the seq: 22  father process output the seq: 11  father process output the seq: 34  father process output the seq: 17  father process output the seq: 52  father process output the seq: 26  father process output the seq: 13  father process output the seq: 40  father process output the seq: 20  father process output the seq: 10  father process output the seq: 5  father process output the seq: 16  father process output the seq: 8  father process output the seq: 4  father process output the seq: 2  father process output the seq: 1 |

#### 测试用例3

|  |
| --- |
| 输入65535 |
| =============lab2 shared memory=============  please input the number: 65535  child process start.  child: 65535  child: 196606  child: 98303  child: 294910  child: 147455  child: 442366  child: 221183  child: 663550  child: 331775  child: 995326  child: 497663  child: 1492990  child: 746495  child: 2239486  child: 1119743  child: 3359230  child: 1679615  child: 5038846  child: 2519423  child: 7558270  child: 3779135  child: 11337406  child: 5668703  child: 17006110  child: 8503055  child: 25509166  child: 12754583  child: 38263750  child: 19131875  child: 57395626  child: 28697813  child: 86093440  child: 43046720  child: 21523360  child: 10761680  child: 5380840  child: 2690420  child: 1345210  child: 672605  child: 2017816  child: 1008908  child: 504454  child: 252227  child: 756682  child: 378341  child: 1135024  child: 567512  child: 283756  child: 141878  child: 70939  child: 212818  child: 106409  child: 319228  child: 159614  child: 79807  child: 239422  child: 119711  child: 359134  child: 179567  child: 538702  child: 269351  child: 808054  child: 404027  child: 1212082  child: 606041  child: 1818124  child: 909062  child: 454531  child: 1363594  child: 681797  child: 2045392  child: 1022696  child: 511348  child: 255674  child: 127837  child: 383512  child: 191756  child: 95878  child: 47939  child: 143818  child: 71909  child: 215728  child: 107864  child: 53932  child: 26966  child: 13483  child: 40450  child: 20225  child: 60676  child: 30338  child: 15169  child: 45508  child: 22754  child: 11377  child: 34132  child: 17066  child: 8533  child: 25600  child: 12800  child: 6400  child: 3200  child: 1600  child: 800  child: 400  child: 200  child: 100  child: 50  child: 25  child: 76  child: 38  child: 19  child: 58  child: 29  child: 88  child: 44  child: 22  child: 11  child: 34  child: 17  child: 52  child: 26  child: 13  child: 40  child: 20  child: 10  child: 5  child: 16  child: 8  child: 4  child: 2  child process has writen data to shared memory.  child process has done, and exit.  seqlength in father process: 131  father process output the seq: 65535  father process output the seq: 196606  father process output the seq: 98303  father process output the seq: 294910  father process output the seq: 147455  father process output the seq: 442366  father process output the seq: 221183  father process output the seq: 663550  father process output the seq: 331775  father process output the seq: 995326  father process output the seq: 497663  father process output the seq: 1492990  father process output the seq: 746495  father process output the seq: 2239486  father process output the seq: 1119743  father process output the seq: 3359230  father process output the seq: 1679615  father process output the seq: 5038846  father process output the seq: 2519423  father process output the seq: 7558270  father process output the seq: 3779135  father process output the seq: 11337406  father process output the seq: 5668703  father process output the seq: 17006110  father process output the seq: 8503055  father process output the seq: 25509166  father process output the seq: 12754583  father process output the seq: 38263750  father process output the seq: 19131875  father process output the seq: 57395626  father process output the seq: 28697813  father process output the seq: 86093440  father process output the seq: 43046720  father process output the seq: 21523360  father process output the seq: 10761680  father process output the seq: 5380840  father process output the seq: 2690420  father process output the seq: 1345210  father process output the seq: 672605  father process output the seq: 2017816  father process output the seq: 1008908  father process output the seq: 504454  father process output the seq: 252227  father process output the seq: 756682  father process output the seq: 378341  father process output the seq: 1135024  father process output the seq: 567512  father process output the seq: 283756  father process output the seq: 141878  father process output the seq: 70939  father process output the seq: 212818  father process output the seq: 106409  father process output the seq: 319228  father process output the seq: 159614  father process output the seq: 79807  father process output the seq: 239422  father process output the seq: 119711  father process output the seq: 359134  father process output the seq: 179567  father process output the seq: 538702  father process output the seq: 269351  father process output the seq: 808054  father process output the seq: 404027  father process output the seq: 1212082  father process output the seq: 606041  father process output the seq: 1818124  father process output the seq: 909062  father process output the seq: 454531  father process output the seq: 1363594  father process output the seq: 681797  father process output the seq: 2045392  father process output the seq: 1022696  father process output the seq: 511348  father process output the seq: 255674  father process output the seq: 127837  father process output the seq: 383512  father process output the seq: 191756  father process output the seq: 95878  father process output the seq: 47939  father process output the seq: 143818  father process output the seq: 71909  father process output the seq: 215728  father process output the seq: 107864  father process output the seq: 53932  father process output the seq: 26966  father process output the seq: 13483  father process output the seq: 40450  father process output the seq: 20225  father process output the seq: 60676  father process output the seq: 30338  father process output the seq: 15169  father process output the seq: 45508  father process output the seq: 22754  father process output the seq: 11377  father process output the seq: 34132  father process output the seq: 17066  father process output the seq: 8533  father process output the seq: 25600  father process output the seq: 12800  father process output the seq: 6400  father process output the seq: 3200  father process output the seq: 1600  father process output the seq: 800  father process output the seq: 400  father process output the seq: 200  father process output the seq: 100  father process output the seq: 50  father process output the seq: 25  father process output the seq: 76  father process output the seq: 38  father process output the seq: 19  father process output the seq: 58  father process output the seq: 29  father process output the seq: 88  father process output the seq: 44  father process output the seq: 22  father process output the seq: 11  father process output the seq: 34  father process output the seq: 17  father process output the seq: 52  father process output the seq: 26  father process output the seq: 13  father process output the seq: 40  father process output the seq: 20  father process output the seq: 10  father process output the seq: 5  father process output the seq: 16  father process output the seq: 8  father process output the seq: 4  father process output the seq: 2  father process output the seq: 1 |

#### 测试用例4

|  |
| --- |
| 输入2021211138 |
| =============lab2 shared memory=============  please input the number: 2021211138  child process start.  child: 2021211138  child: 1010605569  child: -1263150588  child: -631575294  child: -315787647  child: -947362940  child: -473681470  child: -236840735  child: -710522204  child: -355261102  child: -177630551  child: -532891652  child: -266445826  child: -133222913  child: -399668738  child: -199834369  child: -599503106  child: -299751553  child: -899254658  child: -449627329  child: -1348881986  child: -674440993  child: -2023322978  child: -1011661489  child: 1259982830  child: 629991415  child: 1889974246  child: 944987123  child: -1460005926  child: -730002963  child: 2104958408  child: 1052479204  child: 526239602  child: 263119801  child: 789359404  child: 394679702  child: 197339851  child: 592019554  child: 296009777  child: 888029332  child: 444014666  child: 222007333  child: 666022000  child: 333011000  child: 166505500  child: 83252750  child: 41626375  child: 124879126  child: 62439563  child: 187318690  child: 93659345  child: 280978036  child: 140489018  child: 70244509  child: 210733528  child: 105366764  child: 52683382  child: 26341691  child: 79025074  child: 39512537  child: 118537612  child: 59268806  child: 29634403  child: 88903210  child: 44451605  child: 133354816  child: 66677408  child: 33338704  child: 16669352  child: 8334676  child: 4167338  child: 2083669  child: 6251008  child: 3125504  child: 1562752  child: 781376  child: 390688  child: 195344  child: 97672  child: 48836  child: 24418  child: 12209  child: 36628  child: 18314  child: 9157  child: 27472  child: 13736  child: 6868  child: 3434  child: 1717  child: 5152  child: 2576  child: 1288  child: 644  child: 322  child: 161  child: 484  child: 242  child: 121  child: 364  child: 182  child: 91  child: 274  child: 137  child: 412  child: 206  child: 103  child: 310  child: 155  child: 466  child: 233  child: 700  child: 350  child: 175  child: 526  child: 263  child: 790  child: 395  child: 1186  child: 593  child: 1780  child: 890  child: 445  child: 1336  child: 668  child: 334  child: 167  child: 502  child: 251  child: 754  child: 377  child: 1132  child: 566  child: 283  child: 850  child: 425  child: 1276  child: 638  child: 319  child: 958  child: 479  child: 1438  child: 719  child: 2158  child: 1079  child: 3238  child: 1619  child: 4858  child: 2429  child: 7288  child: 3644  child: 1822  child: 911  child: 2734  child: 1367  child: 4102  child: 2051  child: 6154  child: 3077  child: 9232  child: 4616  child: 2308  child: 1154  child: 577  child: 1732  child: 866  child: 433  child: 1300  child: 650  child: 325  child: 976  child: 488  child: 244  child: 122  child: 61  child: 184  child: 92  child: 46  child: 23  child: 70  child: 35  child: 106  child: 53  child: 160  child: 80  child: 40  child: 20  child: 10  child: 5  child: 16  child: 8  child: 4  child: 2  child process has writen data to shared memory.  child process has done, and exit.  seqlength in father process: 194  father process output the seq: 2021211138  father process output the seq: 1010605569  father process output the seq: -1263150588  father process output the seq: -631575294  father process output the seq: -315787647  father process output the seq: -947362940  father process output the seq: -473681470  father process output the seq: -236840735  father process output the seq: -710522204  father process output the seq: -355261102  father process output the seq: -177630551  father process output the seq: -532891652  father process output the seq: -266445826  father process output the seq: -133222913  father process output the seq: -399668738  father process output the seq: -199834369  father process output the seq: -599503106  father process output the seq: -299751553  father process output the seq: -899254658  father process output the seq: -449627329  father process output the seq: -1348881986  father process output the seq: -674440993  father process output the seq: -2023322978  father process output the seq: -1011661489  father process output the seq: 1259982830  father process output the seq: 629991415  father process output the seq: 1889974246  father process output the seq: 944987123  father process output the seq: -1460005926  father process output the seq: -730002963  father process output the seq: 2104958408  father process output the seq: 1052479204  father process output the seq: 526239602  father process output the seq: 263119801  father process output the seq: 789359404  father process output the seq: 394679702  father process output the seq: 197339851  father process output the seq: 592019554  father process output the seq: 296009777  father process output the seq: 888029332  father process output the seq: 444014666  father process output the seq: 222007333  father process output the seq: 666022000  father process output the seq: 333011000  father process output the seq: 166505500  father process output the seq: 83252750  father process output the seq: 41626375  father process output the seq: 124879126  father process output the seq: 62439563  father process output the seq: 187318690  father process output the seq: 93659345  father process output the seq: 280978036  father process output the seq: 140489018  father process output the seq: 70244509  father process output the seq: 210733528  father process output the seq: 105366764  father process output the seq: 52683382  father process output the seq: 26341691  father process output the seq: 79025074  father process output the seq: 39512537  father process output the seq: 118537612  father process output the seq: 59268806  father process output the seq: 29634403  father process output the seq: 88903210  father process output the seq: 44451605  father process output the seq: 133354816  father process output the seq: 66677408  father process output the seq: 33338704  father process output the seq: 16669352  father process output the seq: 8334676  father process output the seq: 4167338  father process output the seq: 2083669  father process output the seq: 6251008  father process output the seq: 3125504  father process output the seq: 1562752  father process output the seq: 781376  father process output the seq: 390688  father process output the seq: 195344  father process output the seq: 97672  father process output the seq: 48836  father process output the seq: 24418  father process output the seq: 12209  father process output the seq: 36628  father process output the seq: 18314  father process output the seq: 9157  father process output the seq: 27472  father process output the seq: 13736  father process output the seq: 6868  father process output the seq: 3434  father process output the seq: 1717  father process output the seq: 5152  father process output the seq: 2576  father process output the seq: 1288  father process output the seq: 644  father process output the seq: 322  father process output the seq: 161  father process output the seq: 484  father process output the seq: 242  father process output the seq: 121  father process output the seq: 364  father process output the seq: 182  father process output the seq: 91  father process output the seq: 274  father process output the seq: 137  father process output the seq: 412  father process output the seq: 206  father process output the seq: 103  father process output the seq: 310  father process output the seq: 155  father process output the seq: 466  father process output the seq: 233  father process output the seq: 700  father process output the seq: 350  father process output the seq: 175  father process output the seq: 526  father process output the seq: 263  father process output the seq: 790  father process output the seq: 395  father process output the seq: 1186  father process output the seq: 593  father process output the seq: 1780  father process output the seq: 890  father process output the seq: 445  father process output the seq: 1336  father process output the seq: 668  father process output the seq: 334  father process output the seq: 167  father process output the seq: 502  father process output the seq: 251  father process output the seq: 754  father process output the seq: 377  father process output the seq: 1132  father process output the seq: 566  father process output the seq: 283  father process output the seq: 850  father process output the seq: 425  father process output the seq: 1276  father process output the seq: 638  father process output the seq: 319  father process output the seq: 958  father process output the seq: 479  father process output the seq: 1438  father process output the seq: 719  father process output the seq: 2158  father process output the seq: 1079  father process output the seq: 3238  father process output the seq: 1619  father process output the seq: 4858  father process output the seq: 2429  father process output the seq: 7288  father process output the seq: 3644  father process output the seq: 1822  father process output the seq: 911  father process output the seq: 2734  father process output the seq: 1367  father process output the seq: 4102  father process output the seq: 2051  father process output the seq: 6154  father process output the seq: 3077  father process output the seq: 9232  father process output the seq: 4616  father process output the seq: 2308  father process output the seq: 1154  father process output the seq: 577  father process output the seq: 1732  father process output the seq: 866  father process output the seq: 433  father process output the seq: 1300  father process output the seq: 650  father process output the seq: 325  father process output the seq: 976  father process output the seq: 488  father process output the seq: 244  father process output the seq: 122  father process output the seq: 61  father process output the seq: 184  father process output the seq: 92  father process output the seq: 46  father process output the seq: 23  father process output the seq: 70  father process output the seq: 35  father process output the seq: 106  father process output the seq: 53  father process output the seq: 160  father process output the seq: 80  father process output the seq: 40  father process output the seq: 20  father process output the seq: 10  father process output the seq: 5  father process output the seq: 16  father process output the seq: 8  father process output the seq: 4  father process output the seq: 2  father process output the seq: 1 |

### 4.6 结果分析

由于程序没有充分的溢出判断，导致在输入为2021211138时序列出现了正溢出和负溢出。

程序开始时，主进程创建一个子进程。父进程和子进程是独立运行的，拥有各自的内存空间。在子进程中，计算Collatz序列的过程在共享内存中执行。子进程开始计算，将每个值存储在共享内存中，并在计算完成后退出。这意味着子进程在其自己的地址空间中执行计算操作，而通过共享内存与父进程通信。父进程使用waitpid等待子进程的退出。这会导致父进程挂起，直到子进程完成计算。这确保父进程只在子进程完成后才会继续执行。子进程在完成计算后退出，并将其退出状态设置为Collatz序列的长度，让父进程知道共享内存中的序列有多长。父进程在子进程完成后，使用mmap访问共享内存中的序列数据，并输出计算得到的Collatz序列，包括长度以及具体的数值。共享内存的强大功能，允许父子进程之间有效地共享数据。共享内存的强大功能使得父进程和子进程可以独立操作，同时访问相同的共享内存区域，这是操作系统提供的一种进程间通信方式。

## 5 实验三

### 5.1 实验三内容

设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一个进程。比如，第一个进程发消息：“I am Here”，第二个进程收到后，将它改变为：“i AM hERE”之后，再发给第一个进程。提示：

（1）需要创建子进程，父子进程之间通过普通管道进行通信。

（2）需要建立两个普通管道。

### 5.2 主要系统调用

1、pipe(pipeName)函数

pipe()，用于创建一个无命名（或匿名）管道。这个系统调用的目的是在父进程和子进程之间建立一个管道，以便它们可以进行进程间通信。

参数 pipeName是一个整型数组，通常包含两个整数。这两个整数代表管道的两个文件描述符：

pipeName[0]：代表管道的读取端，父进程可以从这个文件描述符中读取数据。

pipeName[1]：代表管道的写入端，父进程可以通过这个文件描述符向管道写入数据。

一旦使用 pipe(pipeName) 创建了管道，父进程可以使用 write(pipeName[1], data, size) 将数据写入管道，而子进程可以使用 read(pipeName[0], buffer, size) 从管道读取数据。

2、close(pipeParent[x])

close(pipeParent[x]) 是一个系统调用，用于关闭管道的文件描述符，其中 x 是文件描述符的索引，通常是0或1。管道有两个文件描述符，一个用于读取数据，另一个用于写入数据。

pipeParent[0] 通常是管道的读取端，用于从管道中读取数据。

pipeParent[1] 通常是管道的写入端，用于将数据写入管道。

当你调用 close(pipeParent[x]) 时，它会关闭指定的文件描述符。这是为了告诉操作系统不再使用该文件描述符。在管道通信中，关闭文件描述符通常用于以下目的：

关闭写入端 (close(pipeParent[1]))：在写入数据的进程完成后，通过关闭写入端，它告诉管道不会再有新数据写入。这对于通知读取数据的进程非常重要，它知道什么时候停止等待更多数据。

关闭读取端 (close(pipeParent[0]))：在读取数据的进程完成后，通过关闭读取端，它告诉管道不会再从中读取数据。这可以帮助通知写入数据的进程停止等待数据被读取。

关闭不再需要的文件描述符：如果某个进程已经完成了对特定文件描述符的读取或写入，那么关闭它可以释放相关的资源，从而有助于更好的资源管理。

3、read()

read(pipeParent[0], message, MSG\_SIZE) 是一个系统调用，用于从管道的读取端 (pipeParent[0]) 读取数据并存储到缓冲区 message 中。这个调用通常用于管道通信，其中 pipeParent[0] 代表管道的读取端，message 是用于接收数据的缓冲区，MSG\_SIZE 表示要读取的最大字节数。

pipeParent[0] 是管道的读取端，它用于从管道中读取数据。

message 是一个字符数组或缓冲区，用于存储从管道中读取的数据。

MSG\_SIZE 表示要读取的最大字节数，通常是 message 缓冲区的大小。

read 函数将从管道中读取数据并将其存储到 message 缓冲区中。它会返回实际读取的字节数。通常，你需要检查 read 的返回值，以确保你读取了正确数量的字节。如果 read 返回的字节数小于 MSG\_SIZE，这意味着管道中没有更多的数据可供读取，因此可以根据需要进行处理。

### 5.3 程序设计

程序的主要思路是，父进程负责向子进程发送消息，而子进程负责接收消息并修改大小写后再发送给父进程。这种父子进程之间的通信通过管道实现。程序运行后，用户可以在终端中输入一条消息，然后程序将修改消息的大小写，并最终输出。

定义消息的最大大小

|  |
| --- |
| #define MSG\_SIZE 1024 |

定义两个通信信道

|  |
| --- |
| int pipeParent[2], pipeChild[2];  char message[MSG\_SIZE]; |

子进程处理流程

|  |
| --- |
| if (child\_pid == 0) {  // 子进程 - 接收消息并修改大小写  close(pipeParent[1]); // 关闭父进程管道的写端  // 从父进程管道中读取消息  read(pipeParent[0], message, MSG\_SIZE);  printf("Child process receive the message: %s\n",message);  close(pipeParent[0]);  // 修改消息大小写  for (int i = 0; message[i] != '\0'; i++) {  if (islower(message[i])) {  message[i] = toupper(message[i]);  } else if (isupper(message[i])) {  message[i] = tolower(message[i]);  }  }  // 发送修改后的消息到子线程管道  close(pipeChild[0]); //关闭子线程管道读端  write(pipeChild[1], message, strlen(message) + 1);  printf("Child send the message: %s\n",message);  close(pipeChild[1]);  } |

父进程处理流程

|  |
| --- |
| // 父进程 - 发送消息  close(pipeParent[0]); // 关闭管道1的读端  printf("Enter a message: ");  fgets(message, MSG\_SIZE, stdin);  // 发送消息到管道1  write(pipeParent[1], message, strlen(message) + 1);  close(pipeParent[1]);  close(pipeChild[1]); // 关闭管道2的写端  // 从管道2中读取修改后的消息  read(pipeChild[0], message, MSG\_SIZE);  close(pipeChild[0]);  printf("Received modified message: %s", message); |

### 5.4 程序源代码

|  |
| --- |
| c |
| #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h> #include <ctype.h> #include <sys/tpes.h> #include <sys/mman.h> #include <sys/wait.h>  #define MSG\_SIZE 1024  int main() {     // 0读 1写     int pipeParent[2], pipeChild[2];     char message[MSG\_SIZE];      // 创建两个管道     if (pipe(pipeParent) == -1 || pipe(pipeChild) == -1) {         perror("Pipe creation failed");         exit(1);     }      pid\_t child\_pid = fork();      if (child\_pid == -1) {         perror("Fork failed");         exit(1);     }      if (child\_pid == 0) {         // 子进程 - 接收消息并修改大小写         close(pipeParent[1]); // 关闭父进程管道的写端          // 从父进程管道中读取消息         read(pipeParent[0], message, MSG\_SIZE);         printf("Child process receive the message: %s\n",message);         close(pipeParent[0]);          // 修改消息大小写         for (int i = 0; message[i] != '\0'; i++) {             if (islower(message[i])) {                 message[i] = toupper(message[i]);              } else if (isupper(message[i])) {                 message[i] = tolower(message[i]);             }         }          // 发送修改后的消息到子线程管道         close(pipeChild[0]);    //关闭子线程管道读端         write(pipeChild[1], message, strlen(message) + 1);         printf("Child send the message: %s\n",message);         close(pipeChild[1]);     } else {         // 父进程 - 发送消息         close(pipeParent[0]); // 关闭管道1的读端         printf("Enter a message: ");         fgets(message, MSG\_SIZE, stdin);          // 发送消息到管道1         write(pipeParent[1], message, strlen(message) + 1);         close(pipeParent[1]);         close(pipeChild[1]); // 关闭管道2的写端         // 从管道2中读取修改后的消息         read(pipeChild[0], message, MSG\_SIZE);         close(pipeChild[0]);          printf("Received modified message: %s", message);     }      return 0; } |

### 5.5 测试用例及结果

测试用例1

|  |
| --- |
| 输入hello world |
| Enter a message: hello world  Child process receive the message: hello world  Child send the message: HELLO WORLD  Received modified message: HELLO WORLD |

测试用例2

|  |
| --- |
| 输入Hello World |
| Enter a message: Hello World  Child process receive the message: Hello World  Child send the message: hELLO wORLD  Received modified message: hELLO wORLD |

测试用例3

|  |
| --- |
| 输入  123456asldkfjkasldSDLKFJKALSDKJF098!@#$%^&\*$%^&\*()\_+MBCRFTYUK |
| Enter a message:  123456asldkfjkasldSDLKFJKALSDKJF098!@#$%^&\*$%^&\*()\_+MBCRFTYUK  Child process receive the message:  123456asldkfjkasldSDLKFJKALSDKJF098!@#$%^&\*$%^&\*()\_+MBCRFTYUK  Child send the message:  123456ASLDKFJKASLDsdlkfjkalsdkjf098!@#$%^&\*$%^&\*()\_+mbcrftyuk  Received modified message:  123456ASLDKFJKASLDsdlkfjkalsdkjf098!@#$%^&\*$%^&\*()\_+mbcrftyuk |

### 5.6 结果分析

父进程和子进程分别执行不同的任务，并通过管道进行通信。让我体会到操作系统如何管理多个进程同时运行，每个进程具有自己的独立内存空间和执行流。通过 fork 系统调用，父进程可以创建子进程。这是在操作系统中创建新进程的常见方式。子进程是父进程的副本，但具有独立的地址空间，使它们能够同时运行。使用了两个管道，分别用于父进程到子进程和子进程到父进程的通信。让我知道了操作系统如何通过管道允许进程之间传递数据。关闭不需要的管道端有助于确保数据流正确。父进程和子进程之间通过管道进行通信，以传递消息。这是一种常见的进程间通信（IPC）方法。操作系统负责管理管道的数据传输和同步，以确保正确的数据传递。父进程使用 wait 或 waitpid 系统调用等待子进程完成。操作系统在子进程完成时会通知父进程，父进程能够获取子进程的退出状态，确保进程安全退出。通过运行父进程和子进程并发执行，我知道了操作系统如何在多个进程之间共享处理器时间片，以实现并发执行。