电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2021090922022

姓 名 韩政君

（实验） 课程名称 操作系统原理与实践

理论教师 赵太银

实验教师 冯月

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：韩政君 学号：2021090922022 指导教师：冯月**

**实验地点：信软学院楼西304 实验时间：2023.04.06**

1. **实验名称：**

哲学家就餐问题

**二、实验学时：**4学时

**三、实验目的：**

掌握哲学家就餐问题，掌握预防死锁的实现方法；了解Linux系统下进程和线程的实现；掌握通过互斥量、POSIX信号量、XSI信号量集实现多线程/多进程同步控制的方法。

**四、实验原理：**

哲学家进餐问题描述有五个哲学家，他们的生活方式是交替地进行思考和进餐，哲学家们共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五支筷子，平时哲学家进行思考，饥饿时便试图取其左、右最靠近他的筷子，只有在他拿到两支筷子时才能进餐，进餐完毕，放下筷子又继续思考。

筷子是临界资源，一段时间只允许一位哲学家使用。为了表示互斥，用一个信号量表示一只筷子，五个信号量构成信号量数组。

约束条件

(1)只有拿到两只筷子时，哲学家才能吃饭。

(2)如果筷子已被别人拿走，则必须等别人吃完之后才能拿到筷子。

(3)任一哲学家在自己未拿到两只筷子吃饭前，不会放下手中拿到的筷子。

Semaphore chopstick[5]={1，l，1，l，1}；

p(stick[i])；

p(stick[(i+1) % 5])；

进餐；

v(stick[i])；

v(stick[(i+1) % 5])；

当哲学家饥饿时，总是先去拿他左边的筷子，执行wait(chopstick[I])，成功后，再去拿他右边的筷子，执行wait(chopstick[I+1]%5)；成功后便可进餐。进餐毕，先放下他左边的筷子，然后再放下右边的筷子。当五个哲学家同时去取他左边的筷子，每人拿到一只筷子且不释放，即五个哲学家只得无限等待下去，引起死锁。程序应该用相应措施避免死锁或者在程序发生死锁时解除死锁。

确保同一时间里只有一个线程访问共享资源或临界区域。互斥量（mutex）本质上是一把锁，在访问共享资源后临界区域前，对互斥量进行加锁，在访问完成后释放互斥量上的锁，对互斥量进行加锁后，任何其他试图再次对互斥量加锁的线程将会被阻塞，直到锁被释放。

**五、实验内容：**

熟悉Ubuntu系统环境和命令；熟悉Ubuntu系统下的多线程/多进程编程；在Ubuntu系统下编程实现哲学家就餐问题。实现教材2.5.2节中所描述的哲学家就餐问题。要求显示出每个哲学家的工作状态，如吃饭，思考。连续运行30次以上都未出现死锁现象。

**六、实验器材（设备、元器件）：**

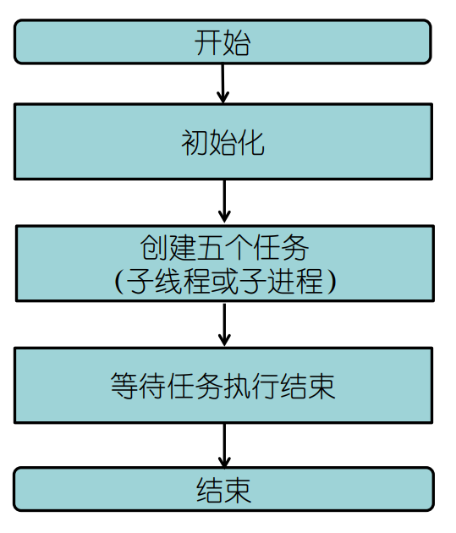
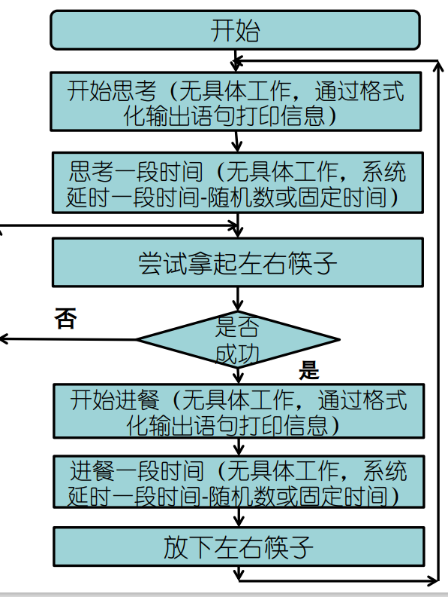
PC计算机，操作系统：Ubuntu

**七、实验步骤：**

1. 使用了Windows的Linux子系统（Ubuntu内核）。

2.使用vscode远程连接wsl进行实验操作

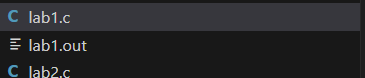
3.程序实现

**** 哲学家函数 主程序

4. 编译、调试、运行

* 命令行下的编译运行

（1）C语言的编译：使用gcc命令（对于使用了线程的，需要加上-lpthread选项），如对lab1文件，执行gcc lab1.c -o lab1 -lpthread，可以看到生成了lab1可执行文件

（2）可执行文件（包括C语言编译生成的文件和sh文件）的执行：如对lab1文件，执行./lab1

5. 获取实验结果并分析

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

哲学家就餐问题程序代码

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t chopsticks[5];

sem\_t room;

int philosopher\_num[5] = {0, 1, 2, 3, 4};

void philosopher(void \*arg)

{

    int id = \*(int \*)arg;

    while (1)

    {

        printf("philosopher %d is thinking\n", id);

        sleep(1);

        while (1)

        {

            sem\_wait(&room);

            sem\_wait(&chopsticks[id]);

            sem\_wait(&chopsticks[(id + 1) % 5]);

            printf("philosopher %d is eating\n", id);

            sleep(1);

            sem\_post(&chopsticks[id]);

            sem\_post(&chopsticks[(id + 1) % 5]);

            sem\_post(&room);

            break;

        }

    }

}

int main()

{

    // 初始化信号量

    sem\_init(&room, 0, 4);

    for (int i = 0; i < 5; i++)

    {

        sem\_init(&chopsticks[i], 0, 1);

    }

    // 创建5个哲学家线程

    pthread\_t philosophers[5];

    for (int i = 0; i < 5; i++)

    {

        pthread\_create(&philosophers[i], NULL, (void \*)philosopher, (void \*)&philosopher\_num[i]);

    }

    // 等待5个哲学家线程结束

    for (int i = 0; i < 5; i++)

    {

        pthread\_join(philosophers[i], NULL);

    }

    // 销毁信号量

    sem\_destroy(&room);

    for (int i = 0; i < 5; i++)

    {

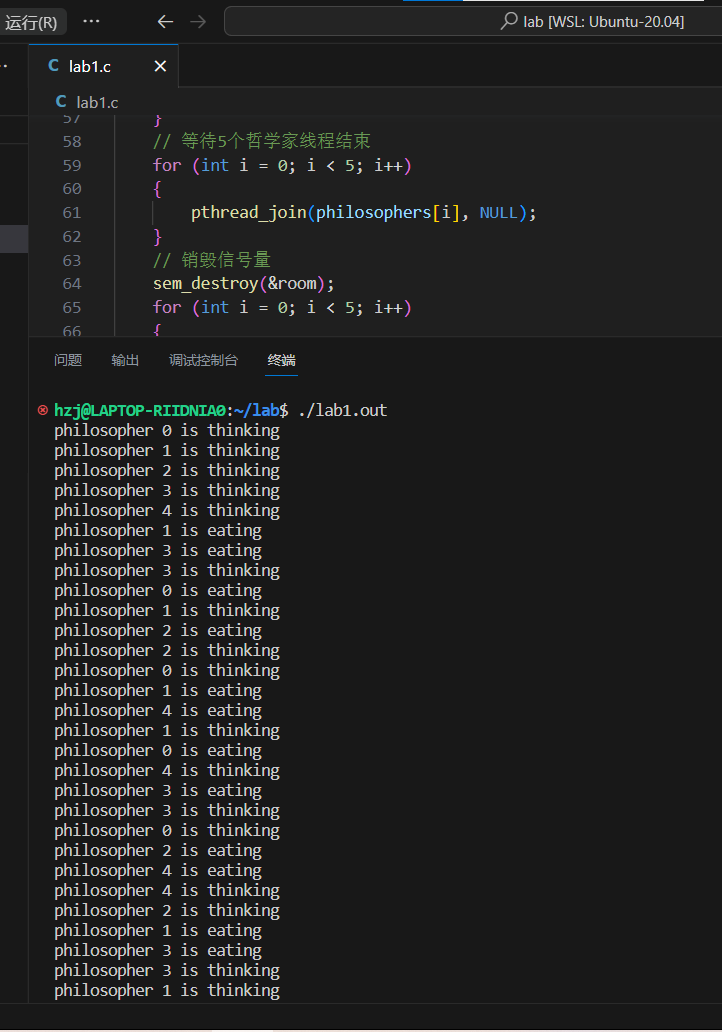
        sem\_destroy(&chopsticks[i]);

    }

    return 0;

}

实验结果输出



**九、总结及心得体会：**

通过这次实验，我加深了对互斥量和信号量的理解，使用了Linux提供的互斥量和信号量定义来解决临界区问题和进程同步问题。在哲学家进餐问题和生产者消费者问题中,我通过信号量实现了线程间的协调,避免了死锁发生。

**十、对本实验过程及方法、手段的改进建议：**

**报告评分：**

**指导教师签字：**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：韩政君 学号：2021090922022 指导教师：冯月**

**实验地点：信软学院楼西304 实验时间：2023.05.10**

**一、实验名称：**生产者和消费者问题

**二、实验学时：**4学时

**三、实验目的：**

掌握进程、线程的概念，熟悉相关的控制语。

掌握进程、线程间的同步原理和方法。

掌握进程、线程间的互斥原理和方法。

掌握使用信号量原语解决进程、线程间互斥和同步方法。

**四、实验原理：**

n个缓冲区的缓冲池作为一个临界资源：当生产者任务从数据源—文件中读取数据后将会申请一个缓冲区，并将此数据放入缓冲区中。消费者任务从一个缓冲区中取走数据，并将其中的内容打印输出。当一个生产者任务正在访问缓冲区时，其他生产者和消费者任务不能访问缓冲区当一个消费者任务正在访问缓冲区时，其他其他生产者和消费者任务不能访问缓冲区使用互斥量实现对缓冲池的互斥访问。

生产者任务在向缓冲池放入数据之前需要判断缓冲池中是否还有空的缓冲区，如果有则向空的缓冲区写入，如果没有则等待消费者任务在从缓冲池读取数据之前需要判断缓冲池中是否有已经写入数据的缓冲区，如果有则读取已经写入数据的缓冲区，如果没有则等待使用两个信号量，一个代表空缓冲区的数量，一个代表已经写入数据的缓冲区数量

操作系统中需要一个地位高于进程的管理者来解决公有资源的使用问题，信号量就是操作系统提供的管理公有资源的有效手段，作为操作系统核心代码执行，不受进程调度的打断，信号量代表可用资源实体的数量。

每个信号量s除一个整数值s.count（计数）外，还有一个进程阻塞队列s.queue，信号量只能通过初始化和两个标准的原语（P操作，V操作）来访问，初始化信号量指定s.count为一个非负整数值，表示空闲资源总数（又称为“资源信号量”），在程序运行中如果s.count若为非负值表示当前的空闲资源数，若为负值其绝对值表示当前等待临界区的进程数。二进制信号量：只允许信号量取0或1值（此时信号量功能和互斥量类似）。

**五、实验内容：**

有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池：生产者进程从文件中读取一个数据，并将它存放到一个缓冲区中； 消费者进程从一个缓冲区中取走数据，并输出此数据。生产者和消费者之间必须保持同步原则：不允许消费者进程到一个空缓冲区去取产品；也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品。

创建3进程（或者线程）作为生产者，4个进程（或者线程）作为消费者。创建一个文件作为数据源，文件中事先写入一些内容作为数据。

生产者和消费者进程（或者线程）都具有相同的优先级。

**六、实验器材（设备、元器件）：**

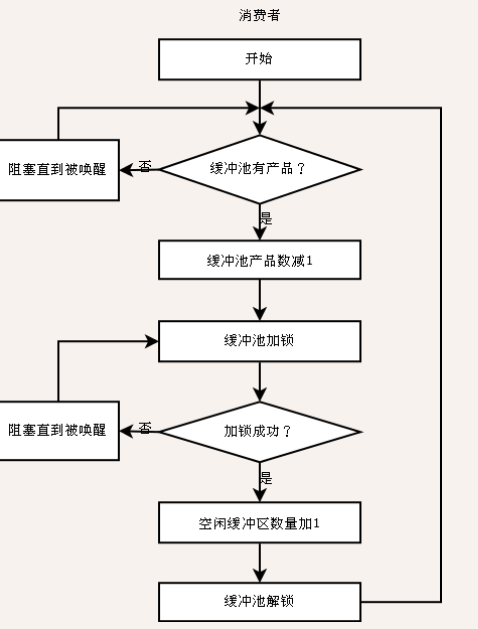
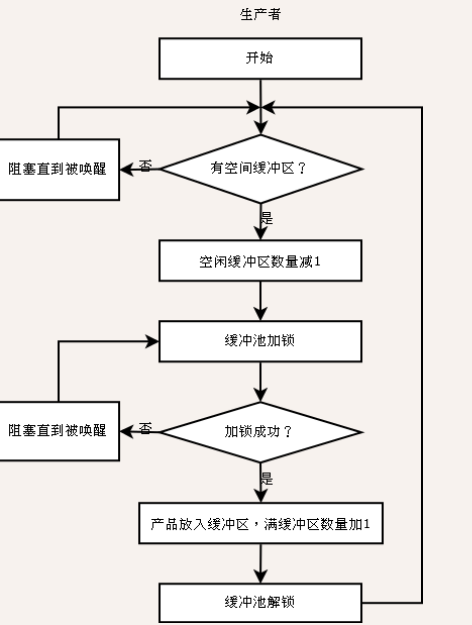
PC计算机，操作系统：Ubuntu

**七、实验步骤：**

1. 使用了Windows的Linux子系统（Ubuntu内核）。

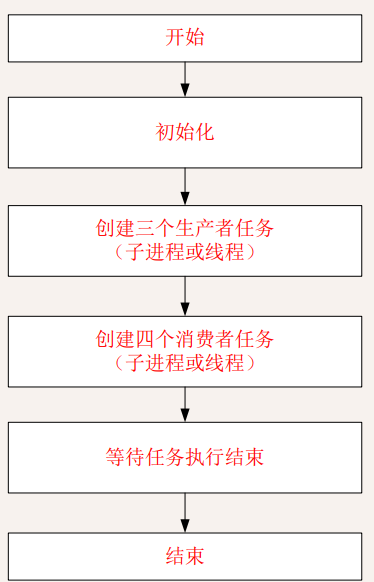
2.使用vscode远程连接wsl进行实验操作

3.程序实现

**** 生产者 消费者

4. 编译、调试、运行

5. 获取实验结果并分析

主程序

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

生产者消费者问题程序代码

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

#include <semaphore.h>

//缓冲区个数

#define N 5

//互斥锁

pthread\_mutex\_t mutex;

//信号量

sem\_t empty;

sem\_t full;

//缓冲区

char buffer[N][1000];

int in=0,out=0;//缓冲区写入、读取

int producer\_num[3]={0,1,2};//生产者

int consumer\_num[4]={0,1,2,3};//消费者

FILE \*fp;

struct paras

{

    int\*num;

    FILE\*fp;

};

void producer(void \*arg)

{

    while(1)

    {

        //参数初始化

        char data[1000];

        struct paras \*p=(struct paras \*)arg;

        int \*num = p->num;

        FILE \*fp = p->fp;

        sem\_wait(&empty);

        pthread\_mutex\_lock(&mutex);

        //读取文件

        memset(data,0,sizeof(data));

        if(fscanf(fp,"%s",data)==EOF)

        {

            fseek(fp,0,SEEK\_SET);

            fscanf(fp,"%s",data);

        }

        //打印信息

        printf("producer %d put %s into buffer[%d]--------------%d\n",\*num,data,in,\*num);

        memset(buffer[in],0,sizeof(buffer[in]));

        strcpy(buffer[in],data);

        //循环队列

        in=(in+1)%N;

        sleep(1);

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&full);

    }

}

void consumer(void \*arg)

{

    while (1)

    {

        // 从一个缓冲区中取走数据

        sem\_wait(&full);

        pthread\_mutex\_lock(&mutex);

        //打印信息

        printf("                                                            consumer %d get %s from buffer[%d]-------------%d\n", \*(int \*)arg, buffer[out], out,\*(int\*)arg);

        out = (out + 1) % N; // 循环队列，out指向下一个存放数据的缓冲区

        sleep(1);            // 为方便调试，每次消费者消费数据后休眠1秒

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&empty);

    }

}

int main()

{

    // 打开文件

    FILE \*fp = fopen("data.txt", "r");

    // 初始化互斥锁

    pthread\_mutex\_init(&mutex, NULL);

    // 初始化信号量

    sem\_init(&empty, 0, N);

    sem\_init(&full, 0, 0);

    // 初始化缓冲区

    for (int i = 0; i < N; i++)

    {

        memset(buffer[i], 0, sizeof(buffer[i]));

    }

    // 创建3个生产者线程

    pthread\_t producers[3];

    for (int i = 0; i < 3; i++)

    {

        struct paras p;

        p.num = &producer\_num[i];

        p.fp = fp;

        pthread\_create(&producers[i], NULL, (void \*)producer, (void \*)&p);

    }

    // 创建4个消费者线程

    pthread\_t consumers[4];

    for (int i = 0; i < 4; i++)

    {

        pthread\_create(&consumers[i], NULL, (void \*)consumer, (void \*)&consumer\_num[i]);

    }

    // 等待3个生产者线程结束

    for (int i = 0; i < 3; i++)

    {

        pthread\_join(producers[i], NULL);

    }

    // 等待4个消费者线程结束

    for (int i = 0; i < 4; i++)

    {

        pthread\_join(consumers[i], NULL);

    }

    // 销毁互斥锁

    pthread\_mutex\_destroy(&mutex);

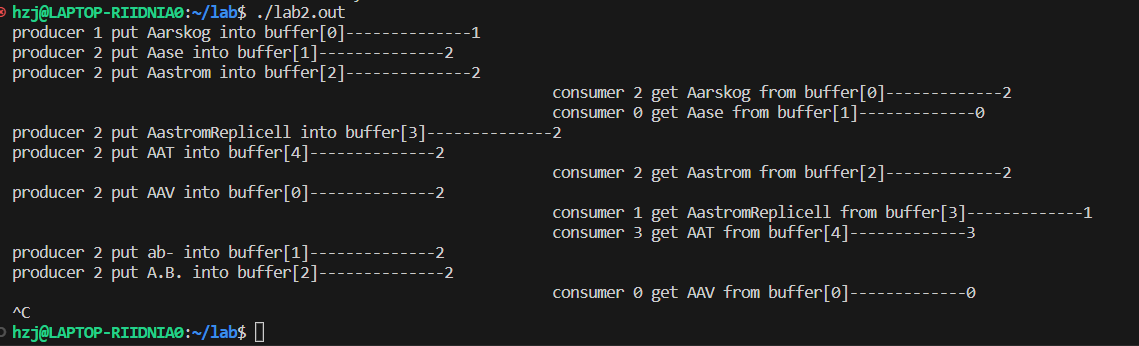
    // 销毁信号量

    sem\_destroy(&empty);

    sem\_destroy(&full);

    return 0;

}

输出结果

**九、总结及心得体会：**

生产者消费者问题是一个经典的同步问题，它涉及到多个线程之间的数据共享和访问，需要使用线程同步和互斥机制来保证数据的正确性和一致性。在Linux编程实验中，我通过实现生产者消费者问题的程序，深刻理解了同步问题的本质和线程同步的技术实现。通过实现生产者消费者问题的程序，我深入理解了线程同步和互斥的技术实现，包括互斥锁、条件变量、信号量等。我还学会了如何使用Linux系统提供的线程库和同步机制来实现多线程编程，并且了解了如何调试和排错多线程程序。

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：韩政君 学号：2021090922022 指导教师：冯月**

**实验地点：信软学院楼西304 实验时间：2023.05.17**

**一、实验名称：**基于管道的进程间数据传输

**二、实验学时：**4学时

**三、实验目的：**

熟悉Linux下的应用程序开发。

熟悉Linux的进程控制原语的使用。

掌握Linux操作系统的进程间通信机制管道的使用。

掌握Linux操作系统中父进程与子进程的同步。

**四、实验原理：**

基于管道的进程间数据传输是一种常见的进程间通信方式，它利用操作系统提供的管道机制，实现了进程间的数据传输。在Linux系统中，管道是一种特殊的文件，它可以实现进程间的单向数据传输。

在基于管道的进程间数据传输中，通常需要创建两个相关的进程，一个是管道的写入进程，一个是管道的读取进程。写入进程向管道中写入数据，读取进程从管道中读取数据。管道的写入进程和读取进程可以是同一个进程，也可以是不同的进程。

管道的创建过程如下：

调用pipe()函数创建一个管道，该函数返回两个文件描述符，一个是读取端文件描述符，一个是写入端文件描述符。

创建一个子进程，在子进程中关闭读取端文件描述符，并将标准输出重定向到写入端文件描述符上。

在父进程中关闭写入端文件描述符，并将标准输入重定向到读取端文件描述符上。

在父进程中向管道中写入数据，子进程从管道中读取数据。

通过上述步骤，父进程和子进程之间就可以实现基于管道的进程间数据传输。

在Linux编程实验中，我通过实现基于管道的进程间数据传输的程序，深刻理解了管道的本质和使用方法。我学会了如何调用pipe()函数创建管道，如何在进程中使用标准输入输出来读取和写入管道中的数据，以及如何使用fork()函数创建子进程，实现进程间的数据传输。

**五、实验内容：**

第一部分：利用管道实现两个进程的通信、

1. 父进程首先使用系统调pipe()建立一个管道，然后使用系统调用fork()创建子进程1，

2. 子进程1关闭管道读文件，

3. 子进程1通过文件I/O操作向管道写文件写一句话（向文件中写入字符串）：Child process 1 is sending a message!

4. 然后子进程1调用exit()结束运行。

5. 父进程再次使用系统调用fork()创建子进程2，

6. 子进程2关闭管道读文件，

7. 子进程2通过文件I/O操作向管道写文件写一句话（向文件中写入字符串）：Child process 2 is sending a message!

8. 然后子进程2调用exit()结束运行。

9. .父进程关闭管道写文件，父进程通过文件I/O操作从管道读文件中读出来自于两个子进程的信息，通过printf语句打印输出在屏幕上。

第二部分：利用消息队列实现进程间的通信

1.编程实现两个进程通过消息队列进行通信，

2.一个server进程，一个client进程。server进程向client进程发送其进程ID，client进程同时也向server进程发送其进程ID，

3.双方接收到消息后，将所接收到的进程ID输出到屏幕上。

**六、实验器材（设备、元器件）：**

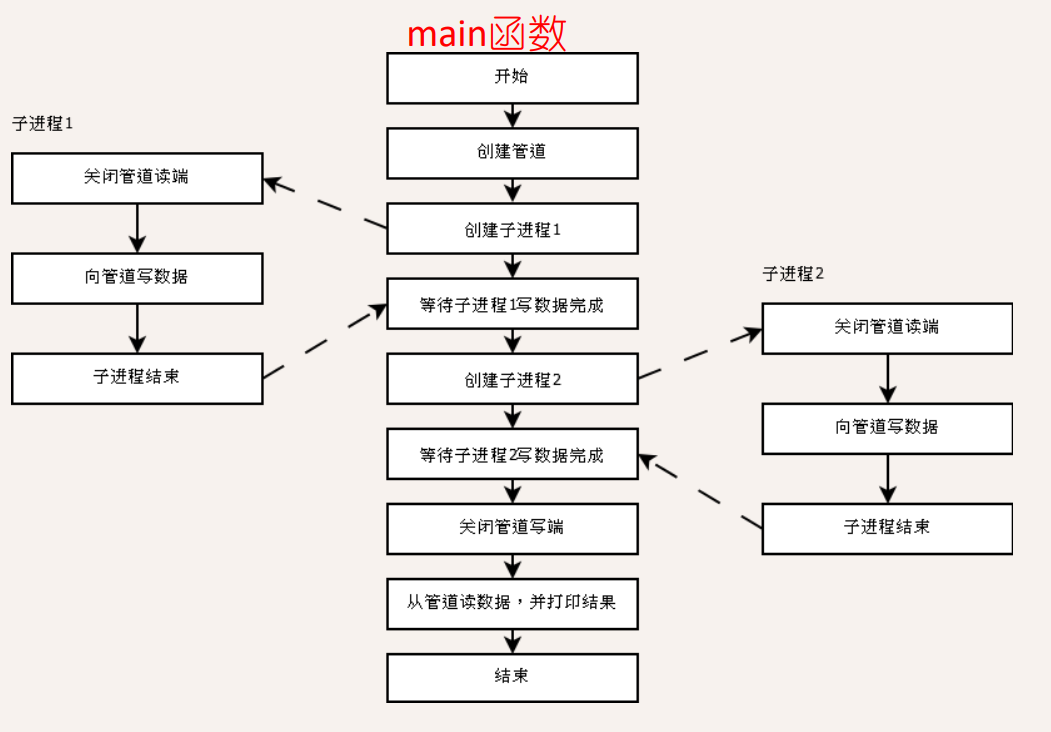
PC计算机，操作系统：Ubuntu

**七、实验步骤：**

1. 使用了Windows的Linux子系统（Ubuntu内核）。

2.使用vscode远程连接wsl进行实验操作

3.程序实现

****

4. 编译、调试、运行

5. 获取实验结果并分析

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

程序代码

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#include <stdlib.h>

#include <sys/wait.h>

#define BUFSIZE 100

#define MAX\_PROCESS\_NUM 2

int process\_num = 0;

int main(void)

{

    char bufin[BUFSIZE] = "empty";

    int bytesin;

    pid\_t childpid[MAX\_PROCESS\_NUM];

    int fd[2];          // fd[0] 读数据, fd[1] 写数据

    if (pipe(fd) == -1) // 创建管道

    {

        perror("Failed to create the pipe");

        return 1;

    }

    bytesin = strlen(bufin);

    for (int i = 0; i < MAX\_PROCESS\_NUM; i++)

    {

        process\_num++; // 子进程标号

        childpid[i] = fork(); // 创建子进程

        switch (childpid[i])

        {

        case -1:

            perror("Failed to fork");

            return 1;

        case 0:

        {

            char bufout[] = "Child process %d is sending a message!\n"; // 子进程写入的字符串

            sprintf(bufout, bufout, process\_num);

            write(fd[1], bufout, strlen(bufout) ); // 写入管道

            printf("Child process %d is sending a message!\n", process\_num);

            exit(0); // 子进程退出

        }

        default:

            // 等待子进程i结束

            waitpid(childpid[i], NULL, 0);

            break;

        }

    }

    printf("Parent process is reading from pipe...\n");

    bytesin = read(fd[0], bufin, BUFSIZE);

    for(int i = 0; i < BUFSIZE; i++)

    {

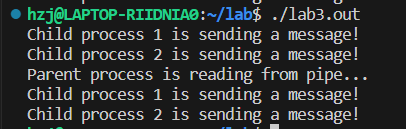
        printf("%c", bufin[i]); // 读取管道，由于字符串在bufin中连续存储，不能直接printf(bufin)，否则只会输出第一个字符串

    }

    return 0;

}

实验结果输出



**九、总结及心得体会：**

在实现基于管道的进程间数据传输的程序过程中，我深刻理解了进程间通信的重要性和管道的原理和使用方法，同时也学会了如何使用Linux系统提供的相关函数来实现基于管道的进程间数据传输。

我认识到进程间通信在多进程编程中的重要性，进程间通信是多进程程序中实现各个进程之间数据交换和协作的关键。基于管道的进程间数据传输是一种常见的进程间通信方式，它可以在不同进程之间传递数据，并且具有较好的可靠性和效率。

**十、对本实验过程及方法、手段的改进建议：**

本次实验使用的是同一个源文件,进程(线程)间通信相对复杂。我建议可以将不同的进程(线程)逻辑写在不同的源文件中,分工合理,利于编码和维护。

**报告评分：**

**指导教师签字：**