**操作系统实验报告**

**学 号\_\_\_\_\_\_\_\_\_18020208\_\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_\_\_丛熙平\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_10/11/2020\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **出勤(5)** | **表现(5)** | | **实验2(15)** | | **实验3(15)** | | **实验4(20)** | **实验5(20)** |
| **□全勤**  **□偶尔缺勤**  **□偶尔出勤**  **□从未出现** | **□积极提问**  **□认真设计**  **□与同学一起商量**  **□主要靠别人讲解**  **□完全不参与** | | **□顺畅且设计合理**  **□执行无问题**  **□有问题，可改正**  **□有问题，无法改正**  **□其他（ ）** | | **□顺畅且设计合理**  **□执行无问题**  **□有问题，可改正**  **□有问题，无法改正**  **□选做题完成+5**  **□其他（ ）** | | **□顺畅且设计合理**  **□执行无问题**  **□有问题，可改正**  **□有问题，无法改正**  **□选做题完成+5**  **□其他（ ）** | **□顺畅且设计合理**  **□执行无问题**  **□有问题，可改正**  **□有问题，无法改正**  **□其他（ ）** |
| **实验6 [选做10]** | | **报告结构(5)** | | **报告内容(15)** | | **教师评语** | | **最终成绩** |
| **□顺畅且设计合理**  **□执行无问题**  **□有问题，可改正**  **□有问题，无法改正**  **□其他（ ）** | | **□完全符合要求**  **□基本符合要求**  **（ ）**  **□有比较多的缺陷**  **（ ）**  **□完全不符合要求**  **（ ）** | | **□充实正确**  **□基本正确**  **（ ）**  **□有一些问题**  **（ ）**  **□问题很大**  **（ ）** | |  | |  |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 目录

[目录 1](#_Toc55974470)

[实验二 2](#_Toc55974471)

[一、 实验目的 2](#_Toc55974472)

[二、 实验内容 2](#_Toc55974473)

[三、 实验设计 2](#_Toc55974474)

[四、 实验结果与分析 4](#_Toc55974475)

[实验三 6](#_Toc55974476)

[一、实验目的 6](#_Toc55974477)

[二、实验内容 6](#_Toc55974478)

[三、实验设计 6](#_Toc55974479)

[四、实验结果与分析 10](#_Toc55974480)

[实验四 11](#_Toc55974481)

[一、实验目的 11](#_Toc55974482)

[二、实验内容 11](#_Toc55974483)

[三、实验设计 12](#_Toc55974484)

[四、实验结果与分析 18](#_Toc55974485)

[实验五 19](#_Toc55974486)

[一、实验目的 19](#_Toc55974487)

[二、实验内容 19](#_Toc55974488)

[三、实验设计 19](#_Toc55974489)

[四、实验结果与分析 22](#_Toc55974490)

[实验六 23](#_Toc55974491)

[一、实验目的 23](#_Toc55974492)

[二、实验内容 23](#_Toc55974493)

[三、实验设计 23](#_Toc55974494)

[四、实验结果与分析 25](#_Toc55974495)

[收获与体会 25](#_Toc55974496)

[参考资料 26](#_Toc55974497)

# 实验二

1. 实验目的

1、加深对进程概念的理解，明确进程与程序的区别；

2、进一步认识并发执行的实质；

3、学习在Linux操作系统中父子进程之间进行管道通信的方法。

1. 实验内容

**1、学习使用fork()系统调用创建子进程，体会父子进程之间的并发关系。**

请编写一段程序，使用系统调用fork()创建两个子进程，实现当此程序运行时，在系统中有一个父进程和两个子进程在活动。父进程的功能是输出一个字符“a”；两个子进程的功能是分别输出一个个字符“b”和一个字符“c”。

多次运行这个程序，试观察记录屏幕上的显示结果，并分析原因。

另外，为了更好地展示进程之间的父子关系，大家可以使用getpid()系统调用来获取当前进程的PID，并用getppid()用于获取当前进程的父进程的PID。

**2、继续体会进程之间的并发关系**

修改刚才的程序，将每一个进程输出一个字符改为用一个循环输出1000个字符（父进程输出1000个“a”，子进程分别输出1000个“b”和“c”），再观察程序执行时屏幕上出现的现象，并分析原因。

**3、进程的管道通信**

编写程序实现进程的管道通信。父进程使用系统调用pipe()创建一个无名管道，二个子进程分别向管道各写一句话：

Child 1 is sending a message!

Child 2 is sending a message!

父进程从管道中读出二个来自子进程的信息并显示出来。

补充材料中给出了管道通信实现过程中需要使用的系统调用的说明，请仔细阅读。

1. 实验设计

Hw2\_1

#include<sys/types.h>

#include<stdio.h>  
#include<unistd.h>

void main()  
{  
 pid\_t pid;  
 pid = fork();  
 if(pid > 0)  
 {  
 pid = fork();  
 if(pid > 0)

{

printf("a, %d\n", getpid());

}

else if(pid == 0)

{

printf("b, %d\n", getpid());

}

}

else if(pid == 0)

{

printf("c, %d\n", getpid());

}

}

Hw2\_2

#include<stdio.h>

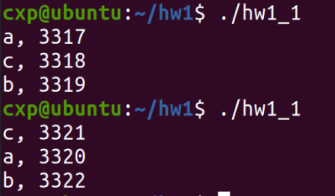
#include<sys/types.h>  
#include<unistd.h>  
  
void temp(char c)  
{  
 for(int i = 0; i < 1000; i++)  
 {  
 printf("%c", c);  
 }  
 printf("\n");  
}  
  
void main()  
{  
 pid\_t pid;  
 pid = fork();  
 if(pid > 0)  
 {  
 pid = fork();  
 if(pid > 0)  
 {  
 temp('a');  
 }  
 else if(pid == 0)  
 {  
 temp('b');  
 }  
 }  
 else if(pid == 0)  
 {  
 temp('c');  
 }  
}

Hw2\_3

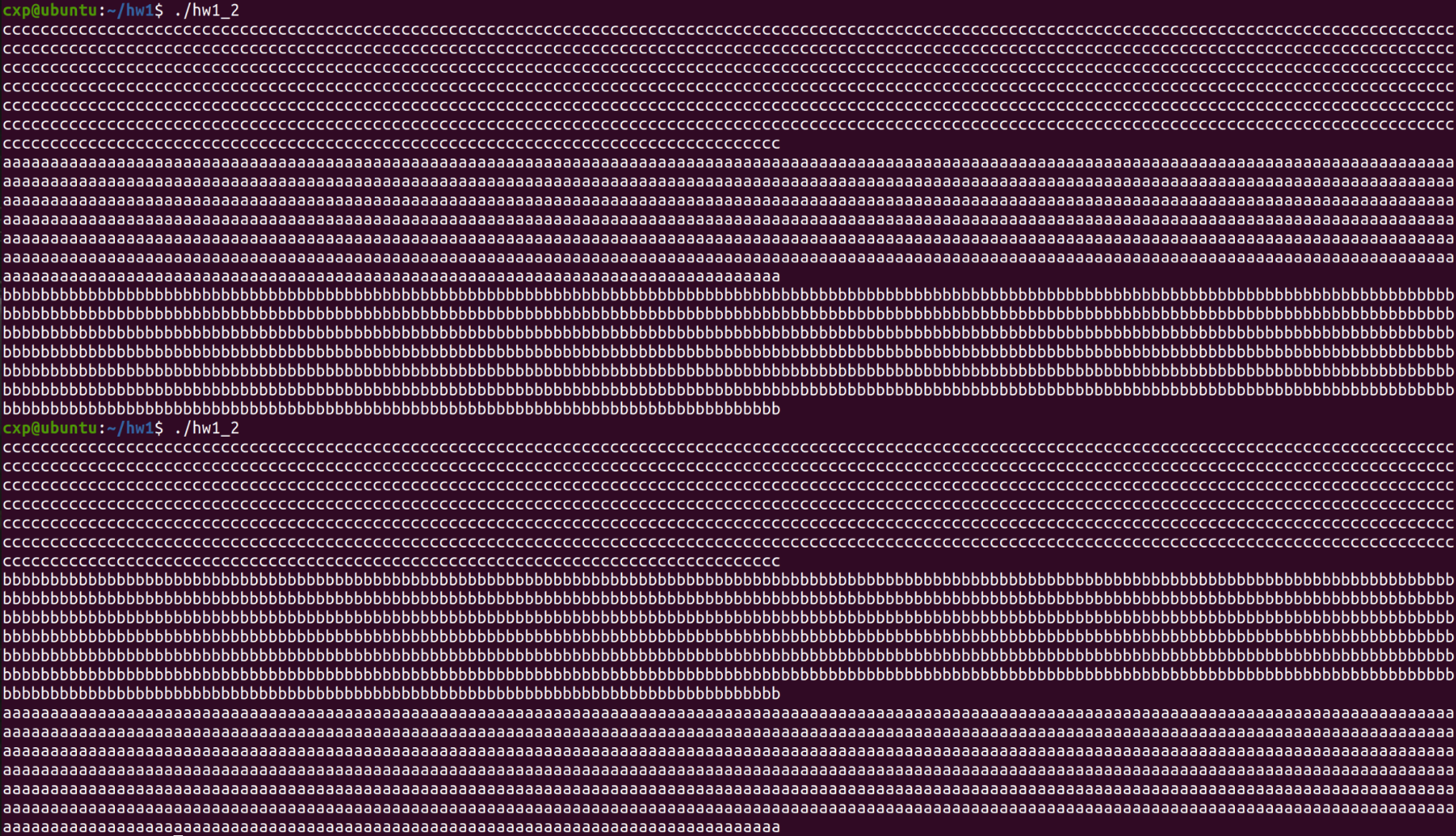
#include<stdio.h>

#include<sys/types.h>  
#include<unistd.h>  
  
void main()  
{  
 pid\_t pid;  
 int fd[2];  
 pipe(fd);  
 pid = fork();  
 if(pid > 0)  
 {  
 pid = fork();  
 if(pid > 0)  
 {  
 wait(NULL);  
 char content[100];  
 read(fd[0], content, 31 \* sizeof(char));  
 printf("%s\n", content);  
 read(fd[0], content, 31 \* sizeof(char));  
 printf("%s\n", content);  
 }  
 else  
 {  
 write(fd[1], "Child 1 is sending a message!\n", 31 \* sizeof(char));  
 }  
 }  
 else  
 {  
 write(fd[1], "Child 2 is sending a message!\n", 31 \* sizeof(char));  
 }  
  
}

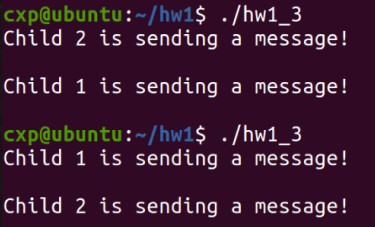
1. 实验结果与分析



运行多次，每次打印的结果顺序应该不太一样，因为父进程没有调用wait()函数，进程调度完全由算法决定



顺序不一样，理由同上



# 实验三

## 一、实验目的

1、编写Linux环境下的多线程程序，了解多线程的程序设计方法，掌握最常用的三个函数pthread\_create，pthread\_join和pthread\_exit的用法；

2、掌握向线程传递参数的方法。

## 二、实验内容

1. **创建线程。**

在这个任务中，需要在主程序中创建两个线程myThread1和myThread2，每个线程打印一句话。**提示：**先定义每个线程的执行体，然后在主函数中使用pthread\_create负责创建两个线程。整个程序等待子线程结束后再退出。

1. **向线程传递参数。**

在上一个程序的基础上，分别向两个线程传递一个字符和一个整数，并让线程负责将两个参数的值打印出来。

1. **【选作题】使用两个线程实现数组排序。**

主程序中用数组data[10]保存10个整数型数据，创建两个线程，一个线程将这个数组中的数据从大到小排列输出；另一个线程将这个数组中的数据从小到大排列输出。

## 三、实验设计

Hw3\_1

#include<stdio.h>

#include<sys/types.h>  
#include<unistd.h>  
#include<pthread.h>  
  
void \*MyThread1(void \*args)  
{  
 printf("This is the thread1 running\n");  
 return (void\*)0;  
}  
  
void \*MyThread2(void \*args)  
{  
 printf("This is the thread2 running\n");  
 return (void\*)0;  
}  
  
void main()  
{  
 int ret1;  
 int ret2;  
 pthread\_t id1;  
 pthread\_t id2;  
 ret1 = pthread\_create(&id1, NULL, (void\*)MyThread1, NULL);  
 ret2 = pthread\_create(&id2, NULL, (void\*)MyThread2, NULL);  
 if(ret1 != 0 || ret2 != 0)  
 {  
 printf("Create Error!\n");  
 }  
 pthread\_join(id1, NULL);  
 pthread\_join(id2, NULL);  
}

Hw3\_2

#include<stdio.h>

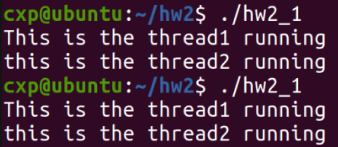
#include<sys/types.h>  
#include<unistd.h>  
#include<pthread.h>  
  
typedef struct information  
{  
 int i;  
 char c;  
}info;  
  
void \*MyThread1(void \*arg)  
{  
 info\* info1;  
 info1 = (info\*)arg;  
 printf("%d %c\n", info1->i, info1->c);  
 return (void\*)0;  
}  
  
void \*MyThread2(void \*arg)  
{  
 info\* info2;  
 info2 = (info\*)arg;  
 printf("%d %c\n", info2->i, info2->c);  
 return (void\*)0;  
}  
  
void main()  
{  
 int ret1, ret2;  
 pthread\_t id1, id2;  
 info info1;  
 info1.i = 1;  
 info1.c = 'o';  
 info info2;  
 info2.i = 2;  
 info2.c = 's';  
 ret1 = pthread\_create(&id1, NULL, MyThread1, (void\*)&info1);  
 ret2 = pthread\_create(&id2, NULL, MyThread2, (void\*)&info2);  
 if(ret1 != 0 || ret2 != 0)  
 {  
 printf("Create Error!\n");  
 }  
 pthread\_join(id1, NULL);  
 pthread\_join(id2, NULL);  
}

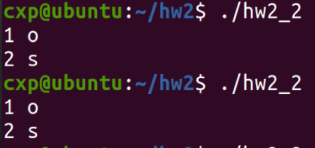
Hw3\_3

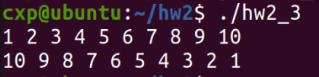
#include<stdio.h>

#include<sys/types.h>  
#include<unistd.h>  
#include<pthread.h>  
  
void \*MyThread1(void\* arg)  
{  
 int\* array1;  
 array1 = (int\*) arg;  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 9; j++)  
 {  
 if(array1[j] > array1[j + 1])  
 {  
 int tmp = array1[j];  
 array1[j] = array1[j + 1];  
 array1[j + 1] = tmp;  
 }  
 }  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 printf("%d ", array1[i]);  
 }  
 printf("\n");  
 return (void\*)0;  
}  
  
void \*MyThread2(void\* arg)  
{  
 int\* array2;  
 array2 = (int\*) arg;  
 for(int i = 0; i < 10; i++)  
 {  
 for(int j = 0; j < 9; j++)  
 {  
 if(array2[j] < array2[j + 1])  
 {  
 int tmp = array2[j];  
 array2[j] = array2[j + 1];  
 array2[j + 1] = tmp;  
 }  
 }  
 }  
 for(int i = 0; i < 10; i++)  
 {  
 printf("%d ", array2[i]);  
 }  
 printf("\n");  
 return (void\*)0;  
}  
void main()  
{  
 int ret1, ret2;  
 pthread\_t id1, id2;  
 int array1[10] = {3, 2, 1, 5, 4, 6, 8, 7, 10, 9};  
 int array2[10] = {3, 2, 1, 5, 4, 6, 8, 7, 10, 9};  
 ret1 = pthread\_create(&id1, NULL, MyThread1, (void\*)array1);  
 ret2 = pthread\_create(&id2, NULL, MyThread2, (void\*)array2);  
 if(ret1 != 0 || ret2 != 0)  
 {  
 printf("Create Error!\n");  
 }  
 pthread\_join(id1, NULL);  
 pthread\_join(id2, NULL);  
  
}

## 四、实验结果与分析







# 实验四

## 一、实验目的

1、学习UNIX类（System V）操作系统信号量机制；

2、编写Linux环境下利用信号量实现进程控制的方法，掌握相关系统调用的使用方法。

## 二、实验内容

**1、生产者消费者问题。**

有数据文件1.dat和2.dat分别存放了10个整数。创建4个线程，其中两个线程read1和read2负责分别从文件1.dat和2.dat中读取一个整数到公共的缓冲区，另两个线程operate1和operate2分别从缓冲区读取数据作加运算和乘运算。使用信号量控制这些线程的执行，保证缓冲区中的数据只能被计算一次（加或者乘），计算完成之后才能继续进行数据的读取工作。

提示：这是一个缓冲区长度为2的生产者消费者问题。read1和read2是生产者，operate1和operate2是消费者。需要互斥使用的缓冲区的长度为2。在进行读取的时候不见得一定是read1读一个数然后read2读一个数，允许read1读得快（或者read2读得快），所以缓冲区中当前的数据都来自于1.dat或者2.dat是被允许的。计算也可能出现连续加和连续乘的情况。

例如，事先编辑好数据文件1.dat和2.dat的内容分别为1 2 3 4 5 6 7 8 9 10和 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10，那么运行你编写的程序，可能得到如下类似的结果：

1+-1=0

-2\*2=-4

3\*4=12

-3\*-4=12

5+-5=0

6\*-6=-36

…

**建议预先写出伪代码描述算法，再进行程序设计实现。**

**2、【选作题】如果严格限制加和乘的两个操作数必须分别来自1.dat和2.dat，且加法和乘法要严格交叉工作，应该如何修改上面的程序？**

**运行结果为：**

1+-1=0

2\*-2=-4

-3+3=0

4\*-4=-16

5+-5=0

-6\*6=-36

…

## 三、实验设计

Hw4\_1

#include<stdio.h>

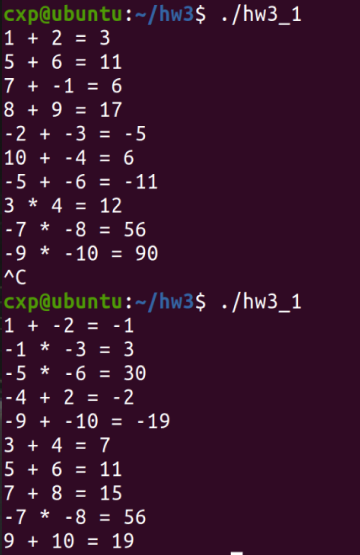
#include<unistd.h>  
#include<sys/types.h>  
#include<pthread.h>  
#include"semaphore.h"  
  
int buffer[2];  
int N = 2;  
sem\_t s, n, e;  
int in = 0;  
int out = 0;  
int end1 = 0, end2 = 0;  
  
int put(int data)  
{  
 buffer[in] = data;  
 in = (++in) % N;  
 return 0;  
}  
  
int take()  
{  
 int res = buffer[out];  
 out = (++out) % N;  
 return res;  
}  
  
void \*MyOperate1(void\* arg)  
{  
 while(1)  
 {  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data1 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data2 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 printf("%d + %d = %d\n", data1, data2, data1 + data2);  
 if(end1 == 1 && end2 == 1) break;  
 }  
 return (void\*) 0;  
}  
  
void \*MyOperate2(void\* arg)  
{  
 while(1)  
 {  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data1 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data2 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 printf("%d \* %d = %d\n", data1, data2, data1 \* data2);  
 if(end1 == 1 && end2 == 1) break;  
 }  
 return (void\*) 0;  
}  
  
void \*MyRead1(void\* arg)  
{  
 FILE\* stream;  
 stream = fopen("/home/cxp/hw3/1.dat", "r");  
 while(1)  
 {  
 int data1;  
 if(fscanf(stream, "%d", &data1) == EOF)  
 {  
 break;  
 }  
 sem\_wait(&e);  
 sem\_wait(&s);  
 put(data1);  
 sem\_post(&s);  
 sem\_post(&n);  
 }  
 end1 = 1;  
}  
  
void \*MyRead2(void\* arg)  
{  
 FILE\* stream;  
 stream = fopen("/home/cxp/hw3/2.dat", "r");  
 while(1)  
 {  
 int data2;  
 if(fscanf(stream, "%d", &data2) == EOF)  
 {  
 break;  
 }  
 sem\_wait(&e);  
 sem\_wait(&s);  
 put(data2);  
 sem\_post(&s);  
 sem\_post(&n);  
 }  
 end2 = 1;  
}  
  
void main()  
{  
 int ret1, ret2, ret3, ret4;  
 pthread\_t op1, op2;  
 pthread\_t read1, read2;  
 sem\_init(&n, 0, 0);  
 sem\_init(&s, 0, 1);  
 sem\_init(&e, 0, 2);  
 ret1 = pthread\_create(&op1, NULL, MyOperate1, NULL);  
 ret2 = pthread\_create(&op2, NULL, MyOperate2, NULL);  
 ret3 = pthread\_create(&read1, NULL, MyRead1, NULL);  
 ret4 = pthread\_create(&read2, NULL, MyRead2, NULL);  
 if(ret1 != 0 || ret2 != 0 || ret3 != 0 || ret4 != 0)  
 {  
 printf("Create Error!\n");  
 }  
 pthread\_join(op1, NULL);  
 pthread\_join(op2, NULL);  
 pthread\_join(read1, NULL);  
 pthread\_join(read2, NULL);  
}

Hw4\_2

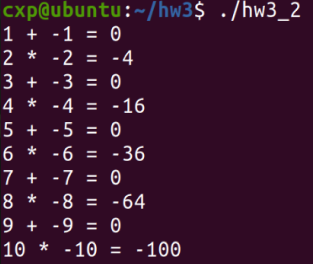
#include<stdio.h>

#include<unistd.h>  
#include<sys/types.h>  
#include<pthread.h>  
#include"semaphore.h"  
  
int buffer[2];  
int N = 2;  
sem\_t s, n, e;  
int in = 0;  
int out = 0;  
int end1 = 0, end2 = 0;  
sem\_t reader1, reader2;  
sem\_t operation1, operation2;  
  
int put(int data)  
{  
 buffer[in] = data;  
 in = (++in) % N;  
 return 0;  
}  
  
int take()  
{  
 int res = buffer[out];  
 out = (++out) % N;  
 return res;  
}  
  
void \*MyOperate1(void\* arg)  
{  
 while(1)  
 {  
 sem\_wait(&operation1);  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data1 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data2 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
 sem\_post(&operation2);  
  
 printf("%d + %d = %d\n", data1, data2, data1 + data2);  
 if(end1 == 1 && end2 == 1) break;  
 }  
 return (void\*) 0;  
}  
  
void \*MyOperate2(void\* arg)  
{  
 while(1)  
 {  
 sem\_wait(&operation2);  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data1 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
  
 sem\_wait(&n);  
 sem\_wait(&s);  
 int data2 = take();  
 sem\_post(&s);  
 sem\_post(&e);  
 sem\_post(&operation1);  
  
 printf("%d \* %d = %d\n", data1, data2, data1 \* data2);  
 if(end1 == 1 && end2 == 1) break;  
 }  
 return (void\*) 0;  
}  
  
void \*MyRead1(void\* arg)  
{  
 FILE\* stream;  
 stream = fopen("/home/cxp/hw3/1.dat", "r");  
 while(1)  
 {  
 int data1;  
 if(fscanf(stream, "%d", &data1) == EOF)  
 {  
 break;  
 }  
 sem\_wait(&reader1);  
 sem\_wait(&e);  
 sem\_wait(&s);  
 put(data1);  
 sem\_post(&s);  
 sem\_post(&n);  
 sem\_post(&reader2);  
 }  
 end1 = 1;  
}  
  
void \*MyRead2(void\* arg)  
{  
 FILE\* stream;  
 stream = fopen("/home/cxp/hw3/2.dat", "r");  
 while(1)  
 {  
 int data2;  
 if(fscanf(stream, "%d", &data2) == EOF)  
 {  
 break;  
 }  
 sem\_wait(&reader2);  
 sem\_wait(&e);  
 sem\_wait(&s);  
 put(data2);  
 sem\_post(&s);  
 sem\_post(&n);  
 sem\_post(&reader1);  
 }  
 end2 = 1;  
}  
  
void main()  
{  
 int ret1, ret2, ret3, ret4;  
 pthread\_t op1, op2;  
 pthread\_t read1, read2;  
 sem\_init(&n, 0, 0);  
 sem\_init(&s, 0, 1);  
 sem\_init(&e, 0, 2);  
 sem\_init(&reader1, 0, 1);  
 sem\_init(&reader2, 0, 0);  
 sem\_init(&operation1, 0, 1);  
 sem\_init(&operation2, 0, 0);  
 ret1 = pthread\_create(&op1, NULL, MyOperate1, NULL);  
 ret2 = pthread\_create(&op2, NULL, MyOperate2, NULL);  
 ret3 = pthread\_create(&read1, NULL, MyRead1, NULL);  
 ret4 = pthread\_create(&read2, NULL, MyRead2, NULL);  
 if(ret1 != 0 || ret2 != 0 || ret3 != 0 || ret4 != 0)  
 {  
 printf("Create Error!\n");  
 }  
 pthread\_join(op1, NULL);  
 pthread\_join(op2, NULL);  
 pthread\_join(read1, NULL);  
 pthread\_join(read2, NULL);  
}

## 四、实验结果与分析



最后有几率出现死锁，因为在两个reader都取完数到文件EOF后，两个operation各取一个数，但是都取不到第二个数，出现死锁，无法执行结束。



# 实验五

## 一、实验目的

1、了解和熟悉Linux支持的消息通信机制及其使用方法

2、了解和熟悉Linux系统的共享存储区的原理及使用方法。

## 二、实验内容

**1、消息的创建、发送和接收**

主函数中使用fork()系统调用创建两个子进程sender和receiver。sender负责接收用户输入的一个字符串，并构造成一条消息，传送给receiver。Receiver在接收到消息后在屏幕上显示出来。此过程一直继续直到用户输入exit为止。在程序设计过程中使用msgget()、msgsnd()、msgrcv()、msgctl()。

**2、共享存储取得创建、附接和断接**

主函数中使用fork()系统调用创建两个子进程sender和receiver。Sender创建共享内存区域并从用户输入得到一个整数放入共享内存区域，receiver负责取出此数并将此整数的平方计算出来。要求程序可以计算10个整数的平方值。在程序设计过程中使用shmget()、shmat()、shmctl()。

## 三、实验设计

Hw5\_1

#include<sys/types.h>

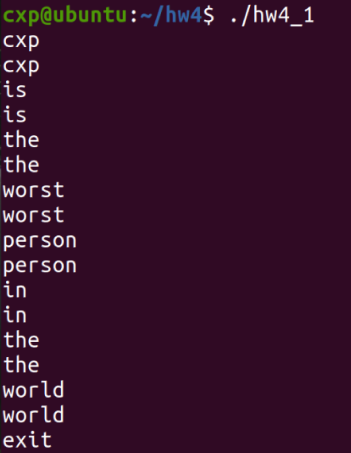
#include<sys/ipc.h>  
#include<sys/msg.h>  
#include<stdio.h>  
#include<unistd.h>  
#include<string.h>  
  
struct message  
{  
 int type;  
 char space[1000];  
};  
  
int open\_queue(int keyval)  
{  
 int qid;  
 qid = msgget(keyval, IPC\_CREAT | 0666);  
 if(qid == -1)  
 {  
 perror("Failed in calling msgget");  
 return (-1);  
 }  
 return (qid);  
}  
  
void main()  
{  
 pid\_t pid;  
 pid = fork();  
 int qid;  
 if(pid > 0)  
 {  
 pid = fork();  
 if(pid > 0)  
 {  
 wait(NULL);  
 }  
 else  
 {  
 qid = open\_queue(1129);  
 struct message msg;  
 while(1)  
 {  
 char input[1000];  
 scanf("%s", input);  
 strcpy(msg.space, input);  
 msgsnd(qid, &msg, 1000, 0);  
 if(input[0] == 'e'&& input[1] == 'x' && input[2] == 'i' && input[3] == 't')  
 {  
 break;  
 }  
 }  
 }  
 }  
 else  
 {  
 qid = open\_queue(1129);  
 struct message msg;  
 while(1)  
 {  
 msgrcv(qid, &msg, 1000, 0, 0);  
 if(msg.space[0] == 'e' && msg.space[1] == 'x' && msg.space[2] == 'i' && msg.space[3] == 't')  
 {  
 break;  
 }  
 printf("%s\n", msg.space);  
 }  
 }  
}

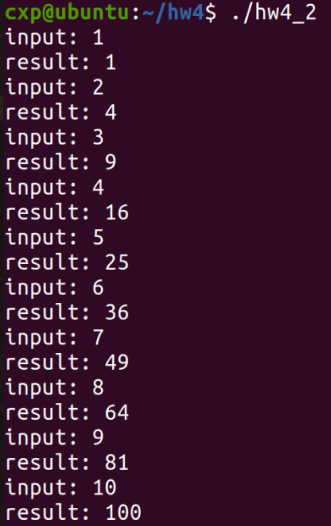
Hw5\_2

#include<stdio.h>

#include<sys/types.h>  
#include<unistd.h>  
#include<sys/ipc.h>  
#include<sys/shm.h>  
#include<math.h>  
#include<string.h>  
  
void main()  
{  
 int pid1, pid2;  
 pid1 = fork();  
 int shmid;  
 int\* addr;  
 int\* data;  
 int key = 1129;  
 if(pid1 > 0)  
 {  
 pid2 = fork();  
 if(pid2 > 0)  
 {  
 wait(pid1);  
 wait(pid2);  
 }  
 else  
 {  
 shmid = shmget(key, 1000, IPC\_CREAT | 0666);  
 addr = shmat(shmid, 0, 0);  
 data = addr + sizeof(int);  
 \*addr = 0;  
 for(int i = 0; i < 10; i++)  
 {  
 printf("input: ");  
 scanf("%d", data);  
 \*addr = 1;  
 while(\*addr != 0)  
 {  
 sleep(0.1);  
 }  
 }  
 }  
 }  
 else  
 {  
 shmid = shmget(key, 1000, IPC\_CREAT | 0666);  
 addr = shmat(shmid, 0, 0);  
 data = addr + sizeof(int);  
 for(int i = 0; i < 10; i++)  
 {  
 while(\*addr != 1)  
 {  
 sleep(0.1);  
 }  
 printf("result: %d\n", (\*data) \* (\*data));  
 \*addr = 0;  
 }  
 shmdt(addr);  
 shmctl(shmid, IPC\_RMID, 0);  
 }  
}

## 四、实验结果与分析





# 实验六

## 一、实验目的

1、学习操作系统信号机制的实现方法；

2、编写Linux环境下利用信号实现进程间通信的方法，掌握注册信号处理程序及信号的发送和接收的一般过程。

## 二、实验内容

1、学习任务。

学习补充材料中的UNIX信号及其描述。

2、使用信号进行进程间通信。

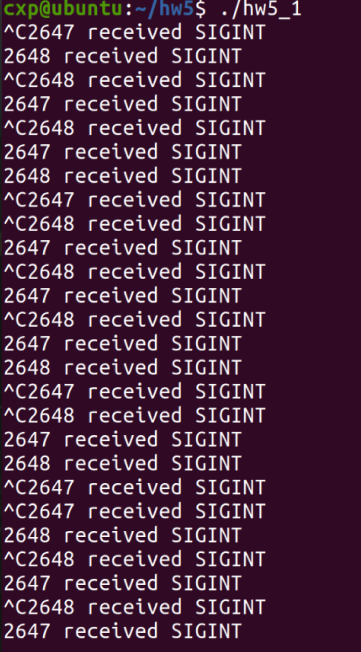
编写一个程序，完成下列功能：实现一个SIGINT、用户自定义信号的处理程序，注册该信号处理程序。主函数中创建一个子进程，令父子进程都进入等待状态。SIGINT、 用户自定义信号的处理程序完成的任务包括：（1）打印接收到的信号的编号（2）打印进程PID。编译并运行该程序，然后在键盘上敲Ctrl + C，观察出现的现象，并解释其含义。

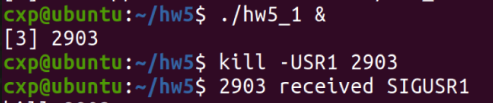
## 三、实验设计

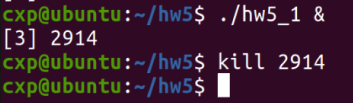
#include<stdio.h>

#include<unistd.h>  
#include<signal.h>  
#include<sys/types.h>  
  
static void sig\_usr(int signo)  
{  
 if(signo == SIGUSR1)  
 {  
 printf("%d received SIGUSR1\n", getpid());  
 }  
 else if(signo == SIGINT)  
 {  
 printf("%d received SIGINT\n", getpid());  
 }  
}  
  
int main()  
{  
 pid\_t pid;  
 pid = fork();  
 if(signal(SIGUSR1, sig\_usr) == SIG\_ERR)  
 {  
 printf("can't catch SIGUSR1\n");  
 return 0;  
 }  
 if(signal(SIGINT, sig\_usr) == SIG\_ERR)  
 {  
 printf("can't catch SIGINT\n");  
 return 0;  
 }  
 for( ; ; ) pause();  
 return 0;  
}

## 四、实验结果与分析







# 收获与体会

通过6次操作系统实验，熟悉了linux指令操作，同时将课上所讲的知识运用到实际中，看到实际运行出的结果更有助于分析结果，加深体会。

# 参考资料

[1]格式请参考操作系统教材的参考资料。左对齐。

[2]

[3]

[4]