

**操作系统**

课程设计实验报告

**题目：** 进程和线程

**专业班级：** 计算机科学与技术 3班

**学 号：** 2017201907

**姓 名：** 赵现锋

**指导教师：** 周翔

**报告日期：** 2019年7月

计算机科学技术学院

**2019年07月**

目录

[一、实验目的 3](#_Toc531100052)

[二、实验题目 3](#_Toc531100053)

[三、设计分析 3](#_Toc531100054)

[四、算法描述](#_Toc531100057) 3

[五、程序](#_Toc531100060) 3

[六、测试与分析 5](#_Toc531100062)

[七、实验总结及体会](#_Toc531100065) 5

[八、参考书目](#_Toc531100066) 5

一、实验目的

1、加深理解进程和程序、进程和线程的联系与区别;

2、深入理解进程及线程的重要数据结构及实现机制;

3、熟悉进程及线程的创建、执行、阻塞、唤醒、终止等控

制方法;

4、学会使用进程及线程开发应用程序。

二、实验题目

设计 – 多线程实现单词统计工具

设有两个文本文件 file1.txt、file2.txt,统计两个文件中单词的总数。

三、设计分析

区分单词原则:凡是一个非字母或数字的字符跟在字母或数字的后面,那么这 个字母或数字就是单词的结尾。

允许线程使用互斥锁来修改临界资源,确保线程间的同步与协作。如果两个 线程需要安全地共享一个公共计数器,需要把公共计数器加锁。线程需要访 问称为互斥锁的变量,它可以使线程间很好地合作,避免对于资源的访问冲突。

四、算法描述

定义一个计数器来统计单词的数量，一个互斥信号量用来对计数器进行互斥访问。编写一个函数用来供线程使用，统计一个文件中单词的个数（注意要使用局部变量）。

五、程序

#include <ctype.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

pthread\_mutex\_t f\_mutex; //互斥信号量

int count = 0; //计数变量

void \*pthread\_function\_countWord(void \*arg) //第一个子线程

{

FILE \*fp; //文件指针

char ch, prech;

char \*fileName = (char \*)arg;

fp = fopen(fileName, "r");

if (!fp) {

perror(fileName);

return NULL;

}

prech = '\0';

while ((ch = fgetc(fp)) != EOF) //以字符为单位读文件

{

if (!isalnum(ch) && isalnum(prech)) {

// TEST

printf("%s WORKING!!! %c %c\n", fileName, ch, prech);

pthread\_mutex\_lock(&f\_mutex);

count++;

pthread\_mutex\_unlock(&f\_mutex);

}

prech = ch;

}

if (isalnum(prech)) {

pthread\_mutex\_lock(&f\_mutex);

count++;

pthread\_mutex\_unlock(&f\_mutex);

}

fclose(fp);

}

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Need two file!\n");

exit(EXIT\_FAILURE);

}

pthread\_t a, b;

int res;

res = pthread\_mutex\_init(&f\_mutex, NULL); //初始化临界区

if (res != 0) {

perror("Mutex initialization failed\n");

exit(EXIT\_FAILURE);

}

//创建第一个子线程

res = pthread\_create(&a, NULL, pthread\_function\_countWord, argv[1]);

if (res != 0) {

perror("Thread1 creation failed\n");

exit(EXIT\_FAILURE);

}

//创建第二个子线程

res = pthread\_create(&b, NULL, pthread\_function\_countWord, argv[2]);

if (res != 0) {

perror("Thread2 creation failed\n");

exit(EXIT\_FAILURE);

}

res = pthread\_join(a, NULL); //等待第一个线程结束

if (res != 0) {

perror("Thread1 join failed\n");

exit(EXIT\_FAILURE);

}

res = pthread\_join(b, NULL); //等待第二个线程结束

if (res != 0) {

perror("Thread2 join failed\n");

exit(EXIT\_FAILURE);

}

printf("There have %d words in two files\n", count); //父线程运行结束

pthread\_mutex\_destroy(&f\_mutex);

return 0;

}

## 六、测试与分析

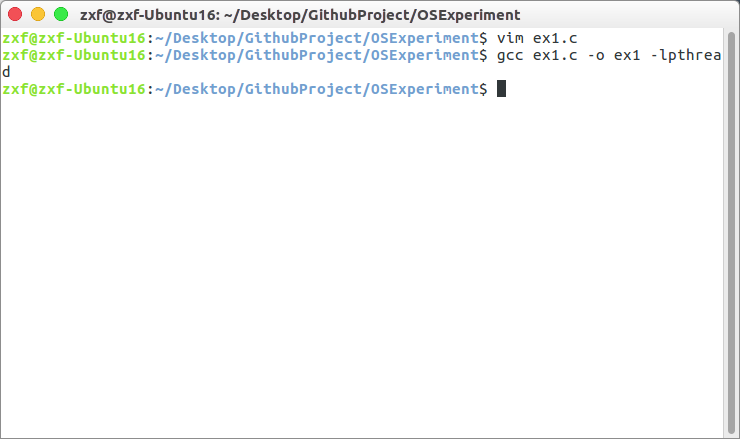
准备好两个记录单词的文本文件1.txt 、2.txt

编写ex1.c文件代码如上：

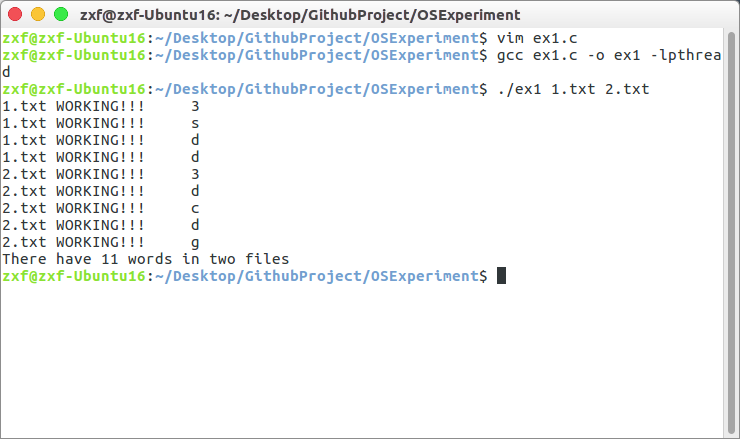




编译.c文件：



运行及结果：



## 七、实验总结及体会

通过此次实验，对进程和线程的概念有了更加深入的理解，对线程之间的互斥和同步有了更加深入的理解，理解了线程对进程资源的继承。 对多线程的代码也有了更加深入的理解，和更加熟练的掌握

## 八、参考书目

《计算机操作系统基础》



**操作系统**

课程设计实验报告

**题目：** 传统的进程间通信

**专业班级：** 计算机科学与技术 3班

**学 号：** 2017201907

**姓 名：** 赵现锋

**指导教师：** 周翔

**报告日期：** 2019年7月

计算机科学技术学院

**2019年07月**

目录

[一、实验目的 10](#_一、实验目的)

[二、实验题目 10](#_二、实验题目)

[三、设计分析 10](#_二、实验题目)

[四、算法描述](#_二、实验题目) 10

[五、程序](#_二、实验题目) 10

[六、测试与分析 11](#_六、测试与分析)

[七、实验总结及体会](#_七、实验总结及体会) 13

[八、参考书目](#_八、参考书目) 13

## 一、实验目的

1、理解信号和管道的概念及用于实现进程通信的原理

2、掌握信号通信机制,实现进程之间通过信号进行通信

3、掌握匿名管道及有名管道通信机制,实现进程之间通过管道进行通信

## 二、实验题目

学会使用有名管道在多进程间建立通信

三、设计分析

为实现多进程间基于命名管道的通信，首先使用mkfifo()创建一个有名管道。之后就可以用传统的文件操作对其进行数据读写，进而完成进程之间的通信。

四、算法描述

先判断管道是否存在，如果不存在则创建管道，如果创建失败则输出错误信息。然后创建子进程，父进程从管道读数据，子进程向管道写数据。完成数据的共享和通信。

五、程序

#include <ctype.h>

#include <errno.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#define FIFO\_SERVER "./fifoserver"

#define BUFFERSIZE 80

int main() {

int fd, ret;

if (access(FIFO\_SERVER, F\_OK) == -1) {

if (mkfifo(FIFO\_SERVER, 0666) < 0 && (errno != EEXIST)) {

printf("Create fifoserver error\n");

exit(0);

}

}

printf("Create fifoserver successfully\n");

pid\_t pid;

pid = fork();

if (pid == 0) {

char bufw[BUFFERSIZE];

printf("Please input a string：");

gets(bufw);

fd = open(FIFO\_SERVER, O\_WRONLY);

if (fd < 0) {

printf("child: Open fifoserver error\n");

exit(0);

}

ret = write(fd, bufw, strlen(bufw));

if (ret < 0) {

printf("fifoserver write error\n");

exit(0);

}

printf("fifoserver write successfully\n");

close(fd);

} else if (pid > 0) {

char bufr[BUFFERSIZE];

memset(bufr, 0, sizeof(bufr));

fd = open(FIFO\_SERVER, O\_RDONLY);

if (fd < 0) {

printf("parent: Open fifoserver error\n");

exit(0);

}

printf("Before: %s\n", bufr);

ret = read(fd, bufr, BUFFERSIZE);

if (ret < 0) {

printf("fifoserver read error\n");

exit(0);

}

printf("After: %s\n", bufr);

close(fd);

} else if (pid < 0) {

printf("Fork error\n");

exit(0);

}

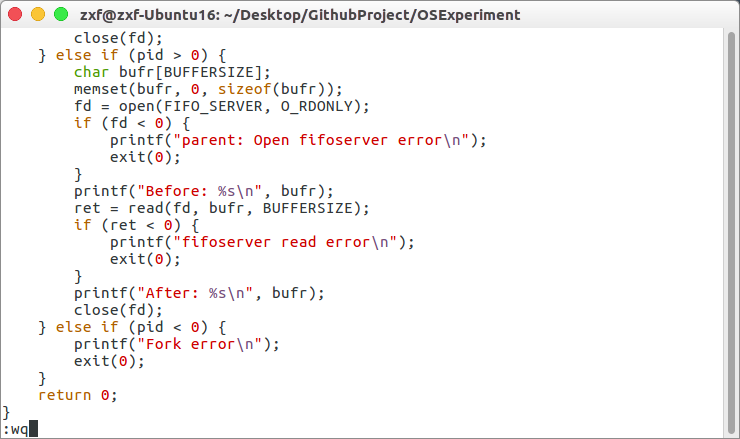
return 0;

}

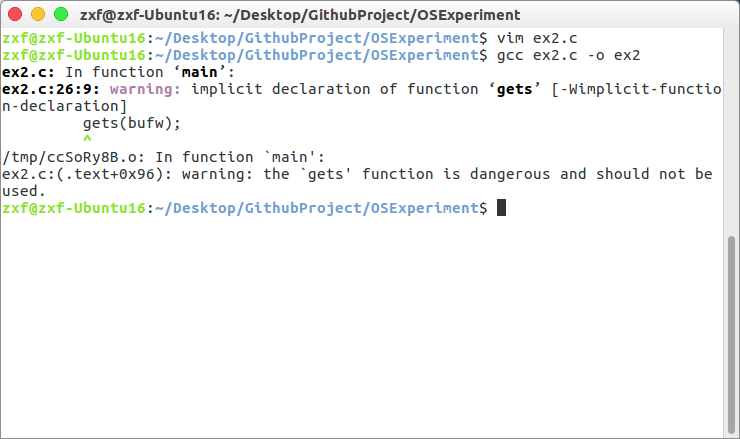
## 六、测试与分析

编写ex2.c 代码如上：

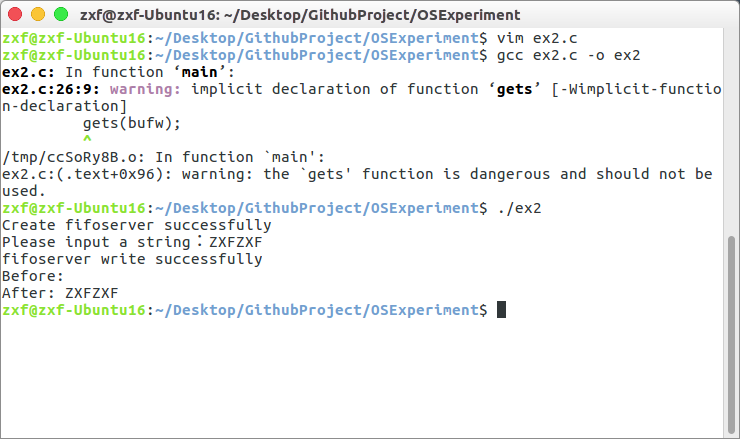




编译ex2.c：



运行及结果：



## 七、实验总结及体会

通过本次实验，对使用管道进行通信有了更加深入的理解，对多进程之间的通信方式有了更多的了解。对管道的概念有了更加深入的思考。

## 八、参考书目

《计算机操作系统基础》



**操作系统**

课程设计实验报告

**题目：** 信号量实现进程同步

**专业班级：** 计算机科学与技术 3班

**学 号：** 2017201907

**姓 名：** 赵现锋

**指导教师：** 周翔

**报告日期：** 2019年7月

计算机科学技术学院

**2019年07月**

目录

[一、实验目的 16](#_一、实验目的_1)

[二、实验题目 16](#_二、实验题目_1)

[三、设计分析 16](#_二、实验题目_1)

[四、算法描述](#_二、实验题目_1) 16

[五、程序](#_二、实验题目_1) 16

[六、测试与分析](#_六、测试与分析_1) 19

[七、实验总结及体会](#_七、实验总结及体会_1) 21

[八、参考书目](#_八、参考书目_1) 21

## 一、实验目的

1、理解 System V IPC 通信机制工作原理;

2、掌握和使用共享主存实现进程通信;

3、掌握和使用消息队列实现进程通信;

4、掌握和使用信号量实现进程同步。

## 二、实验题目

利用信号量解决生产者-消费者问题。（信号量实现进程同步）

三、设计分析

进程同步是操作系统多进程/多线程并发执行的关键之一, 进程同步指为完成共同任务的并发进程基于某个条件来协调它们的活动, 这是进程之间发生的一种直接制约关系, 生产者-消费者问题是典型的进程同步问题, 其本质是如何控制并发进程对有界共享主存区的访问。生产者进程生产产品, 然后将产品放置在一个空缓冲区中供消费者进程消费。 消费者进程从缓冲区中获得产品, 然后释放缓冲区。当生产者进程生产产品时, 如果没有空缓冲区可用, 那么生产者进程必须等待消费者进程释放出一个空缓冲区。 当消费者进程消费产品时,如果没有满的缓冲区, 那么消费者进程将被阻塞, 直到新的产品被生产出来。

四、算法描述

以生产者进程不断向数组添加数据(写入 100 次), 消费者从数组读取数据并求和为例,给出基于信号量解决生产者-消费者问题的程序框架。 该程序假设有一个生产者进程和两个消费者进程, 创建了 fullid、 emptyid 和 mutexid 共 3 个信号量, 供进程间同步访问临界区。 同时, 还建立 4 个共享主存区, 其中 array 用于维护生产者、消费者进程之间的共享数据, sum 保存当前求和结果, 而 set 和 get 分别记录当前生产者进程和消费者进程的读写次数。

五、程序

#include <errno.h>

#include <fcntl.h>

#include <linux/sem.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <time.h>

#include <unistd.h>

#define MAXSEM 5

//声明三个信号灯 ID

int fullid;

int emptyid;

int mutxid;

int main() {

struct sembuf P, V;

union semun arg;

//声明共享主存

int \*array;

int \*sum;

int \*set;

int \*get;

//映射共享主存

array = (int \*)mmap(NULL, sizeof(int) \* MAXSEM, PROT\_READ | PROT\_WRITE,

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

sum = (int \*)mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE,

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

get = (int \*)mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE,

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

set = (int \*)mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE,

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

\*sum = 0;

\*get = 0;

\*set = 0;

//生成信号灯

fullid = semget(IPC\_PRIVATE, 1, IPC\_CREAT | 00666);

emptyid = semget(IPC\_PRIVATE, 1, IPC\_CREAT | 00666);

mutxid = semget(IPC\_PRIVATE, 1, IPC\_CREAT | 00666);

//为信号灯赋值

arg.val = 0;

if (semctl(fullid, 0, SETVAL, arg) == -1) perror("semctl setval error");

arg.val = MAXSEM;

if (semctl(emptyid, 0, SETVAL, arg) == -1) perror("semctl setval error");

arg.val = 1;

if (semctl(mutxid, 0, SETVAL, arg) == -1) perror("setctl setval error");

//初始化 P,V 操作

V.sem\_num = 0;

V.sem\_op = 1;

V.sem\_flg = SEM\_UNDO;

P.sem\_num = 0;

P.sem\_op = -1;

P.sem\_flg = SEM\_UNDO;

//生产者进程

if (fork() == 0) {

int i = 0;

while (i < 100) {

semop(emptyid, &P, 1);

semop(mutxid, &P, 1);

array[\*(set) % MAXSEM] = i + 1;

printf("Producer %d\n", array[(\*set) % MAXSEM]);

(\*set)++;

semop(mutxid, &V, 1);

semop(fullid, &V, 1);

i++;

}

sleep(15);

printf("Producer is over\n");

exit(0);

} else {

// ConsumerA 进程

if (fork() == 0) {

while (1) {

if (\*get == 100) break;

semop(fullid, &P, 1);

semop(mutxid, &P, 1);

\*sum += array[(\*get) % MAXSEM];

printf("The ComsumerA Get Number %d\n", array[(\*get) % MAXSEM]);

(\*get)++;

if (\*get == 100) printf("The sum is %d \n ", \*sum);

semop(mutxid, &V, 1);

semop(emptyid, &V, 1);

sleep(1);

}

printf("ConsumerA is over\n");

exit(0);

} else {

// Consumer B 进程

while (1) {

if (\*get == 100) break;

semop(fullid, &P, 1);

semop(mutxid, &P, 1);

\*sum += array[(\*get) % MAXSEM];

printf("The ComsumerB Get Number %d\n", array[(\*get) % MAXSEM]);

(\*get)++;

if (\*get == 100) printf("The sum is %d \n ", \*sum);

semop(mutxid, &V, 1);

semop(emptyid, &V, 1);

sleep(1);

}

printf("ConsumerB is over\n");

exit(0);

}

}

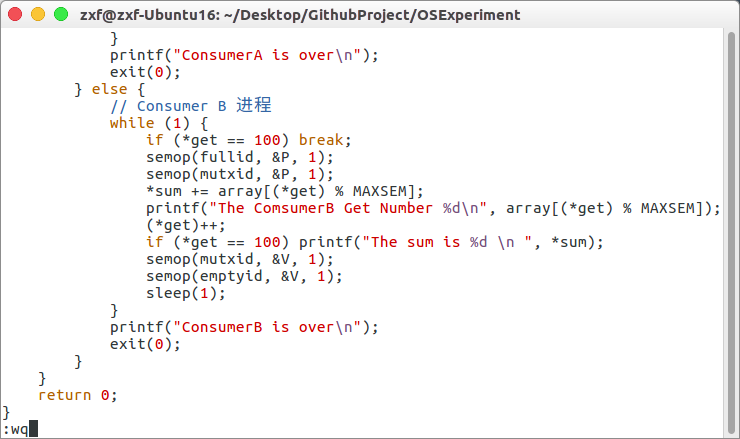
return 0;

}

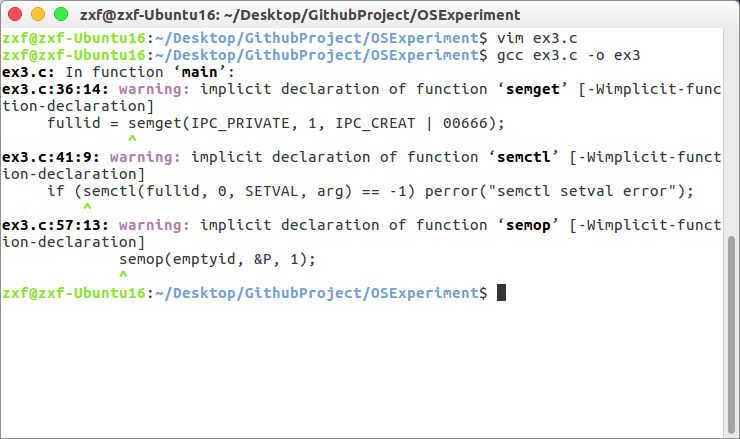
## 六、测试与分析

编写ex3.c文件：

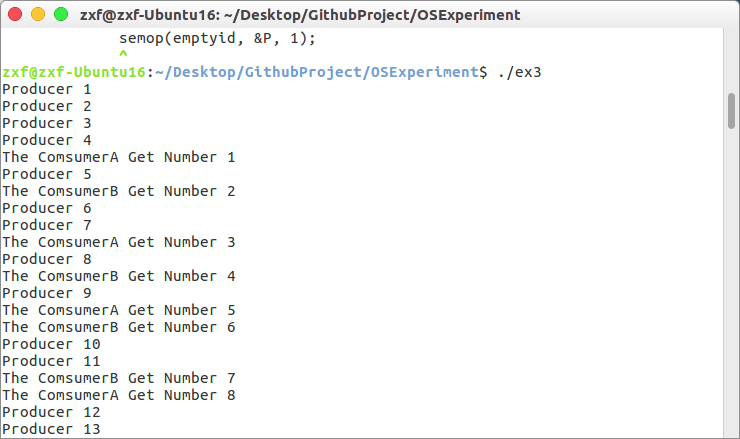


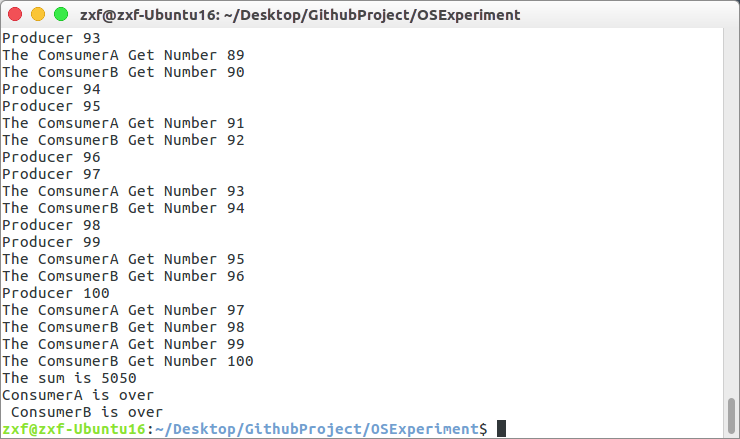


编译ex3.c文件：



运行过程及结果：





## 七、实验总结及体会

通过本次实验，对信号量的代码有了更加深入的理解，对于信号量有了更加深入的思考。理解了应该如何编写代码才可以尽量不发生死锁。理解了System V IPC 通信机制的基本工作原理，并且使用共享主存实现了进程通信。同时熟悉了消息队列实现进程通信。

## 八、参考书目

《计算机操作系统基础》