

**操作系统**

课程设计报告

**题目：** 进程和线程

**专业班级：** 计算机科学与技术 3班

**学 号：** 2017201940

**姓 名：** 魏子钧

**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**

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

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

定义一个互斥信号量，一个共享计数器 。编写一个统计某个文件中单词个 数的函数，分别创建两个线程调用该函数，用互斥信号量控制对计数器的互斥访问.

五、程序

#include <stdio.h>

#include <pthread.h>

#include <ctype.h>

#include <stdlib.h>

//互斥

pthread\_mutexattr\_t counter\_clock;

//计数器

int num = 0;

//线程函数，统计一个文件的单词个数

void \*count(void \*ff){

char \*filename = (char \*)ff;

FILE \*fp;

int c;

int prevc = '\0';

if((fp=fopen(filename,"r"))!=NULL){

## while((c=getc(fp))!=EOF){

## if(!isalnum(c)&&isalnum(prevc)){

## pthread\_mutex\_lock(&counter\_clock);//线程加锁

## num++;

## pthread\_mutex\_unlock(&counter\_clock);//线程解锁

## }

## prevc = c;

## }

## fclose(fp);

## }else{

## perror(filename);

## }

## return NULL;

## }

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

## {

## void \*count\_words(void\*);

## if(argc!=3){//格式错误提示

## printf("Wrong!\n");

## exit(1);

## }

## pthread\_t t1,t2;//两个线程

## int res;

## res = pthread\_mutex\_init(&counter\_clock,NULL);//初始化

## if(res!=0){

## perror("Mutex initialization failed\n");

## exit(EXIT\_FAILURE);

## }

## res = pthread\_create(&t1,NULL,count,argv[1]);//t1创建

## printf("Thread 01 creating...\n");

## if(res!=0){

## perror("Thread create failly\n");

## exit(EXIT\_FAILURE);

## }

## res = pthread\_create(&t2,NULL,count,argv[2]);//t2创建

## printf("Thread 02 creating...\n");

## if(res!=0){

## perror("Thread create failly\n");

## exit(EXIT\_FAILURE);

## }

## res = pthread\_join(t1,NULL);//等t1结束

## printf("Thread 01 joinning...\n");

## if(res!=0){

## perror("Thread join failly\n");

## exit(EXIT\_FAILURE);

## }

## res = pthread\_join(t2,NULL);//等t2结束

## printf("Thread 02 joinning...\n");

## if(res!=0){

## perror("Thread join failly\n");

## exit(EXIT\_FAILURE);

## }

## printf("There are totally %d words\n",num);//输出结果

## pthread\_mutex\_destroy(&counter\_clock);

## return 0;

## }

## 六、测试与分析

### 事先编写好两个文本文件：

### 

### 

### 

### 编译.c文件：

### 

### 运行及结果：

### 

## 七、实验总结及体会

通过此次实验，我更进一步理解了进程和线程间的区别和联系，深入了解了进程和线程的数据结构和实现方法。熟悉了进程和线程的创建、执行、阻塞、唤醒、终止等基本操作。

八、参考书目



**操作系统**

课程设计报告

**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：** 2017201940

**姓 名：** 魏子钧

**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**

## 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()创建一个命名管 道。随后可使用一般的文件I/O函数，如 open(),close(),read(),write()等， 来对它进行操作，从而实现多进程间的通信。

四、算法描述

先创建命名管道，若已有或者创建失败则输出错误信息，否则提示创建成功。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。

五、程序

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <ctype.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#define FIFO\_SERVER "/tmp/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("创建fifoserver失败\n");

exit(0);

}

}

printf("创建fifoserver成功\n");

pid\_t pid;

pid=fork();

### if(pid==0)

### {

### char bufw[BUFFERSIZE];

### printf("请输入：");

### gets(bufw);

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

### if(fd<0)

### {

### printf("child:打开fifoserver失败\n");

### exit(0);

### }

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

### if(ret<0)

### {

### printf("fifoserver写入出错\n");

### exit(0);

### }

### printf("fifoserver写入成功\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:打开fifoserver失败\n");

### exit(0);

### }

### printf("读之前: %s\n",bufr);

### ret=read(fd,bufr,BUFFERSIZE);

### if(ret<0)

### {

### printf("fifoserver读出错\n");

### exit(0);

### }

### printf("读之后: %s\n",bufr);

### close(fd);

### }

### else if(pid<0)

### {

### printf("子进程创建失败\n");

### exit(0);

### }

### return 0;

### }

## 六、测试与分析

## 编写好b.c文件后编译：

### 

### 运行及结果：

### 

## 七、实验总结及体会

通过本次实验，我理解了信号和管道的概念以及用于实现进程通信的原理，同时对信号通信机制有了新的认识。还练习了匿名管道及有名管道通信机制,实现了进程之间通过管道进行通信

## 八、参考书目



**操作系统**

课程设计报告

**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：** 2017201940

**姓 名：** 魏子钧

**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**

## 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

信号量实现进程同步

利用信号量解决生产者-消费者问题。

三、设计分析

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。

四、算法描述

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。

五、程序

#include <sys/mman.h>

#include <sys/types.h>

#include <linux/sem.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <time.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(10);

### printf("Producer is over");

### 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)

### {

### semop(fullid, &P, 1);

### semop(mutxid, &P, 1);

### if(\*get == 100)

### break;

### \*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

### 编写好c.c文件后编译：

### 

### 运行过程及结果：

### 

## 

## 七、实验总结及体会

通过本次实验，我理解了System V IPC 通信机制的基本工作原理,并且和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和信号量实现进程同步。

## 八、参考书目