

**操作系统实验报告**

|  |  |
| --- | --- |
| 姓 名： | 邹雅 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | ACM1901 |
| 学 号： | U201915035 |
| 指导教师： | 张杰 |

2022年 1月 2日

目 录

1 实验一 1

1.1 实验目的 1

1.2 实验内容 1

1.3 实验过程 1

1.4 实验心得 4

附录 实验代码 5

# 实验三

## 实验目的

1、掌握Linux下共享内存的概念与使用方法；

2、掌握环形缓冲的结构与使用方法；

3、掌握Linux下进程同步与通信的主要机制。

## 实验内容

利用多个共享内存（有限空间）构成的环形缓冲，将源文件复制到目标文件，实现两个进程的誊抄。

## 实验过程

## 总体设计

程序主要分为三个部分，且分别对应三个进程：父进程和两个子进程——读进程和写进程。

父进程负责初始化环形缓冲区，创建和初始化信号灯，创建子进程、执行相对应的文件并等待。

写进程负责创建和打开目标文件output.txt，从环形缓冲中读取内容并且写入目标文件。

读进程负责打开输入源文件input.txt，将输入源文件内容写入环形缓冲。

实验三的总体设计如下图所示：

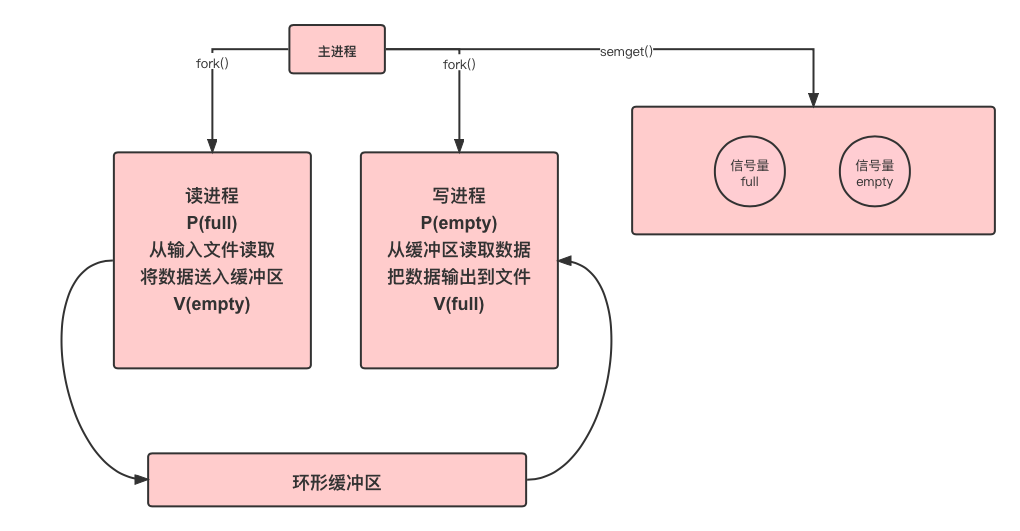


图1-1 总体设计图

## 父进程具体设计

信号灯的操作是由Linux系统实现的。使用semget()函数，使用不同的key创建不同的信号灯，而使用相同的key时创建或者得到的都为同一组信号灯。

父进程创建两个信号灯，创建一个有两个元素的信号灯数组。0号信号灯初始量为0，表示缓冲区已经填入并允许读出；1号信号灯初始量为缓冲区的个数，表示缓冲区空余位置并允许写入的。

接着父进程创建并初始化共享缓冲区。

然后父进程通过fork()创建两个子进程，分别用来执行读文件和写文件两个程序。设置好子进程之后，通过wait(&pid)函数来等待子进程执行完毕。

最后销毁缓冲区，销毁信号量，结束。

## 缓冲区具体设计

共享缓冲区是通过一个结构来进行定义的：

struct share\_buffer{

int flag;

char buf[4096];

};

str为4096个字符构成的数组，也就是4KB的容量，为一个缓冲区能容纳的字符。flag变量用来标记这个共享缓冲区是否对应到文件尾，如果flag为1代表文件正常读入，而如果flag为0则表示此时该缓冲区已经没有输入到了文件末尾。

在主进程创建共享缓冲区的时候设置的大小为sizeof(struct share\_buffer)\*256，也就是总共设置了1MB的缓冲。

## 读文件具体设计

读文件程序先要以读的方式打开“input.txt”，然后得到对应信号灯和共享缓冲区。接着P(full)，缓冲区容量减去1。通过fgets函数从文件指针中得到数据并写入对应的缓冲区：如果还能得到数据，则把该缓冲区的flag设置为1，并且更新缓冲区的index；如果得不到数据了，则把该缓冲区的flag设置为0，并且退出循环。最后V(empty)，增加一个已写入的缓冲区。

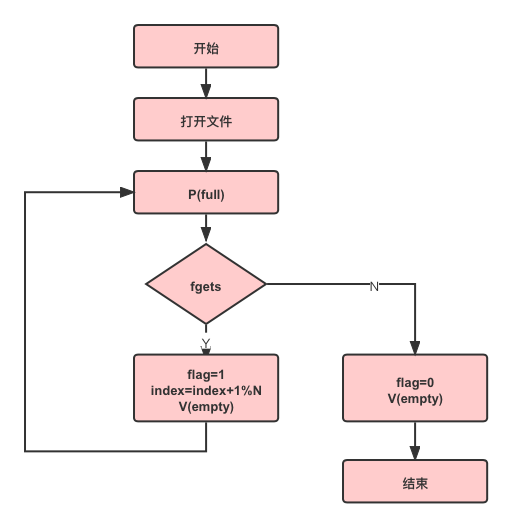


图1-2 读进程流程图

## 写文件具体设计

写文件程序先要以写的方式打开“output.txt”，然后得到对应信号灯和共享缓冲区。接着P(empty)，消耗一个已经写入的缓冲区。检查当前index对应的缓冲区的flag，如果flag为0则退出循环，表示文件已经写完了；如果flag为1，则用fputs函数将缓冲区的内容写入文件，并更新index，最后V(full)，增加一个可写入的缓冲区。

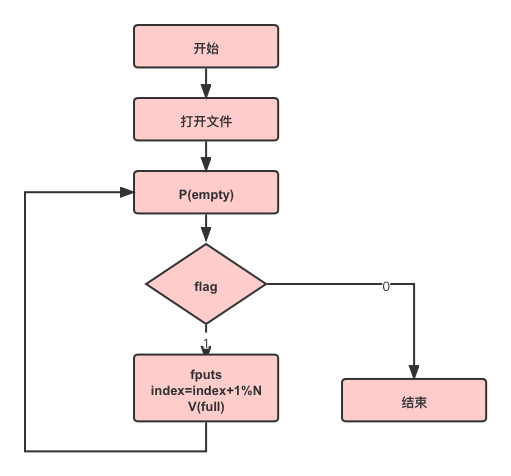


图1-3 写文件流程图

## 开发环境

硬件环境：MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports)

虚拟机操作系统：Ubuntu 16.04.6

编译环境：gcc 5.4.0

代码环境：Vim、gedit

## 调试测试

输入文件属性如下图所示，文件大小为20MB：

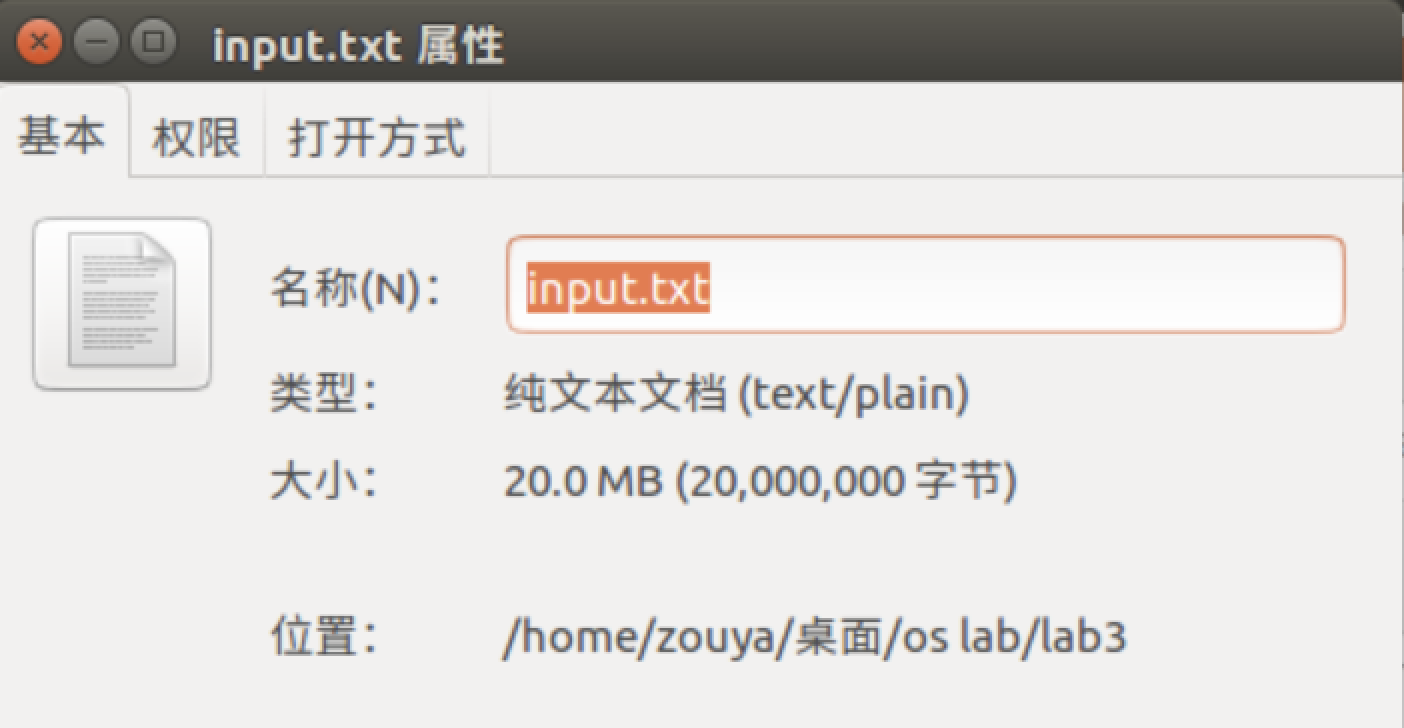


图1-4 输入文件属性

利用gcc编译main.c、input.c和output.c三个文件后，运行程序，结果如下：

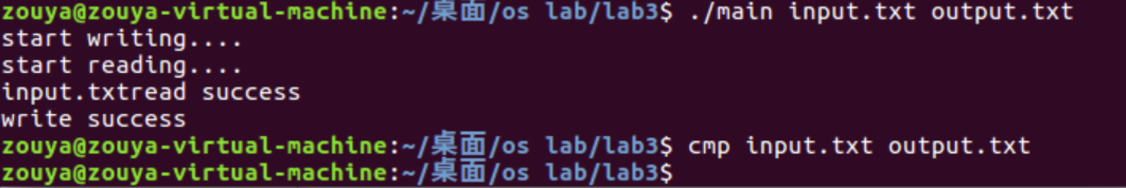


图1-5 运行结果

可以看到利用cmp指令进行比对两个文件，两个文件是一致的。

接着我们利用time指令来查看程序运行时间，结果如下所示：

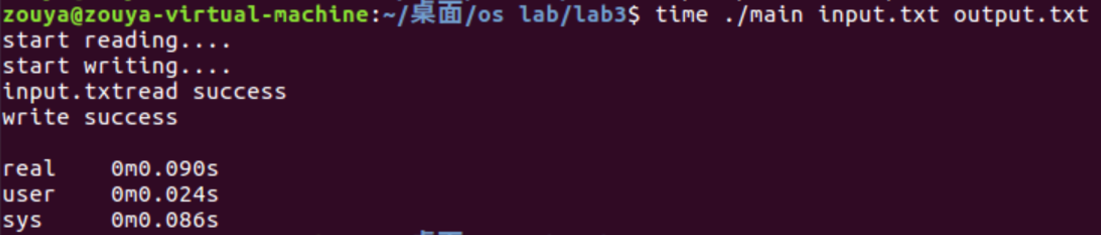


图1-6 运行时间

可以发现，因为我们的共享缓冲区总共有1MB，所以运行相对而言速度较快。

## 实验心得

实验三相比实验一和二要复杂了很多，我在做实验的时候花在实验三的时间也明显多很多。主要是涉及到了信号灯和环形缓冲区。

共享内存区域可以被两个进程同时看到。具体实现上，当两个进程通过页表将虚拟地址映射到物理地址时，在物理地址中有一块共同的内存区，就是共享内存。当一个进程进行写操作而另一个进程进行读操作，就可以实现进程间通信。但是我们要确保一个进程在写的时候不能被读，因此我们使用信号灯来实现同步与互斥。

通过实验也熟练了共享内存 API 的使用。在实验的过程中，尝试使用过链表型的共享内存区域，但是没有调试成功。在失败过多次之后，决定改回数组型的共享内存。其次遇到的问题是怎么控制拷贝结束，最终决定设计一个结构体，以标志位的形式来判定文件尾。在拷贝完毕后，多发送一个带有结束标记的块进入缓冲区，当写进程接收到这一块时就终止，就解决了如何判断程序读到文件尾这个问题。

## 附录 实验代码

1. **header.h 头文件**

#include<sys/types.h>

#include<stdio.h>

#include<pthread.h>

#include<sys/ipc.h>

#include<sys/sem.h>

#include<sys/shm.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<stdlib.h>

#include<string.h>

#define SEMKEY 0x25

#define SHMKEY 0x45

#define BUFNUM 256

#define BUFSIZE 4096

static struct share\_buffer {

int flag;

char buf[BUFSIZE];

};

void P(int semid,int index);

void V(int semid,int index);

void P(int semid,int index){

struct sembuf sem;

sem.sem\_num=index;

sem.sem\_op=-1;

sem.sem\_flg=0;

semop(semid,&sem,1);

}

void V(int semid,int index){

struct sembuf sem;

sem.sem\_num=index;

sem.sem\_op=1;

sem.sem\_flg=0;

semop(semid,&sem,1);

}

1. **main.c**

#include "header.h"

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

int semid = semget(SEMKEY, 2, IPC\_CREAT|0666);

if (semctl(semid, 0, SETVAL, 0) == -1 ||

semctl(semid, 1, SETVAL, BUFNUM) == -1 ){

printf(“semctl initialize error”);

exit(1);

}

int shmid = shmget(SHMKEY, sizeof(struct share\_buffer)\* BUFNUM, IPC\_CREAT|0666);

pid\_t pid1,pid2;

if((pid1=fork())==0){

printf("start reading\n");

execl("./read", "read",argv[1],NULL);

}

else{

if((pid2=fork())==0){

printf("start writing\n");

execl("./write","write",argv[2], NULL);

}

}

int status;

pid1 = wait(&status);

pid2 = wait(&status);

if(semctl(semid, 0, IPC\_RMID, 0)<0){

printf("semctl rmid error!\n");

}

shmctl(shmid, IPC\_RMID, 0);

return 0;

}

1. **read.c**

#include "header.h"

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

printf("%s",argv[1]);

FILE \*fd1 = fopen("input.txt", "r");

int index = 0;

int semid = semget(SEMKEY, 2, 0666); //共享信号灯

int shmid = shmget(SHMKEY, sizeof(struct share\_buffer)\* BUFNUM, 0666);

struct share\_buffer \* buffer= (struct share\_buffer \*)shmat(shmid,NULL,0);

while(1){

P(semid, 1); //共享缓冲区-1

memset(buffer[index].buf, 0, BUFSIZE);

if(fgets(buffer[index].buf, BUFSIZE, fd1)){

buffer[index].flag = 1;

index = (index+1) % BUFNUM;

V(semid,0);

}

else{

buffer[index].flag = 0;

V(semid,0);

break;

}

}

printf("read success\n");

fclose(fd1);

}

1. **write.c**

#include "header.h"

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

FILE \*fd2 = fopen("output.txt", "w");

int index = 0;

int semid = semget(SEMKEY, 2, 0666); //信号灯

int shmid = shmget(SHMKEY, sizeof(struct share\_buffer)\* BUFNUM, 0666);

struct share\_buffer \* buffer = ( struct share\_buffer \*)shmat(shmid, NULL, 0);

while(1){

P(semid, 0);

if(!buffer[index].flag){

break;

}

fputs(buffer[index].buf, fd2);

index = (index+1) % BUFNUM;

V(semid, 1);

}

printf("write success\n");

fclose(fd2);

}