

**课 程 实 验 报 告**

**课程名称： 操 作 系 统 原 理**



**专业班级： 计算机科学与技术校交1701班**

**学 号： U201713666**

**姓 名： 蒋为**

**指导教师： 谢美意**

**报告日期： 2020年1月1日**

**计算机科学与技术学院**

目录

[实验三：共享内存与进程同步 4](#_Toc27558158)

[一、 实验目的 4](#_Toc27558159)

[二、 实验内容 4](#_Toc27558160)

[**1.** **程序要求** 4](#_Toc27558161)

[**2.** **运行环境** 4](#_Toc27558162)

[**3.** **源程序** 4](#_Toc27558163)

[**4.** **实验结果** 11](#_Toc27558164)

[三、 实验心得 12](#_Toc27558165)

# 实验三：共享内存与进程同步

## 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

1. **程序要求**

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。

1. **运行环境**

**软件配置：**

主机：Macos Catalina10.15.1

虚拟机：Paralles Desktop

开发环境：Ubuntu 18.0.1

**硬件：**

处理器：2.3GHz 四核Intel Core i5

内存：16GB 2133MHz LPDDR3

图形卡： Intel Iris Plus Graphics 655 1536MB

1. **源程序**

首先是主进程程序：

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/sem.h>

#include <unistd.h>

#define number 100 //缓冲区的个数

#define size 1024

struct shared\_memory{

char text[number][size]; //一共100个共享内存

};

struct inoutflag{

int in,out,length,flag;

};

int semid; //两个信号，信号0表示空闲缓冲区或者已经被读过的缓冲区个数，信号1表示可用于读取的数据个数

void\* buf;

shared\_memory \*shared;

void\* inouttemp;

inoutflag \*rw;

int read\_proc,write\_proc;

void P(int semid,int index);

void V(int semid,int index);

int main() {

int shm = shmget(1, sizeof(shared\_memory), IPC\_CREAT | 0666);

if (shm == -1) {

printf("共享内存1创建失败！\n");

exit(0);

}

buf=shmat(shm,NULL,SHM\_R|SHM\_W);

shared=(shared\_memory\*)buf;

int shm2=shmget(2,sizeof(inoutflag),IPC\_CREAT|0666);

if(shm2==-1)

{

printf("共享内存2创建失败！\n");

exit(0);

}

inouttemp=shmat(shm2,NULL,SHM\_R|SHM\_W);

rw=(inoutflag\*)inouttemp;

rw->in=0;

rw->out=0;

rw->length=0;

rw->flag=0;

semid = semget(3,2,IPC\_CREAT|0666); //创建信号灯集

semctl(semid,0,SETVAL,100); //0号灯表示空闲缓冲区个数，当前为number

semctl(semid,1,SETVAL,0); //1号灯表示可读取的缓冲区个数，当前为0

if((read\_proc=fork())==0)

{

execv("./read",NULL);

}

else

{

if((write\_proc=fork())==0)

{

execv("./write",NULL);

}

wait(0);

semctl(semid,0,IPC\_RMID);

shmctl(shm,IPC\_RMID,0);

shmctl(shm2,IPC\_RMID,0);

}

return 0;

}

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);

}

然后是read进程程序：

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/sem.h>

#include <unistd.h>

#define number 100 //缓冲区的个数

#define size 1024

struct shared\_memory{

char text[number][size]; //一共100个共享内存

};

struct inoutflag{

int in,out,length,flag;

};

int semid; //两个信号，信号0表示空闲缓冲区或者已经被读过的缓冲区个数，信号1表示可用于读取的数据个数

void\* buf;

shared\_memory \*shared;

void\* inouttemp;

inoutflag \*rw;

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);

}

int main()

{

int shm = shmget(1, sizeof(shared\_memory), IPC\_CREAT | 0666);

if (shm == -1) {

printf("共享内存1创建失败！\n");

exit(0);

}

buf=shmat(shm,NULL,SHM\_R|SHM\_W);

shared=(shared\_memory\*)buf;

int shm2=shmget(2,sizeof(inoutflag),IPC\_CREAT|0666);

if(shm2==-1)

{

printf("共享内存2创建失败！\n");

exit(0);

}

inouttemp=shmat(shm2,NULL,SHM\_R|SHM\_W);

rw=(inoutflag\*)inouttemp;

rw->in=0;

rw->out=0;

rw->length=0;

rw->flag=0;

semid = semget(3,2,IPC\_CREAT|0666); //创建信号灯集

semctl(semid,0,SETVAL,100); //0号灯表示空闲缓冲区个数，当前为number

semctl(semid,1,SETVAL,0); //1号灯表示可读取的缓冲区个数，当前为0

FILE\* rtxt = fopen("/Users/wei/CLionProjects/wei/input.txt","r");

while(1)

{

P(semid,0); //可使用的缓冲区减1

printf("read进程读入缓冲区%d\n",rw->in);

rw->length=fread(shared->text[rw->in], sizeof(char),size,rtxt);

rw->in=(rw->in+1)%number;

if(rw->length!=size)

{

rw->flag=1;

V(semid,1); //可读的缓冲区加1

break;

}

V(semid,1); //可读的缓冲区加1

}

}

最后是write进程程序：

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/sem.h>

#include <unistd.h>

#define number 100 //缓冲区的个数

#define size 1024

struct shared\_memory{

char text[number][size]; //一共100个共享内存

};

struct inoutflag{

int in,out,length,flag;

};

int semid; //两个信号，信号0表示空闲缓冲区或者已经被读过的缓冲区个数，信号1表示可用于读取的数据个数

void\* buf;

shared\_memory \*shared;

void\* inouttemp;

inoutflag \*rw;

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);

}

int main()

{

int shm = shmget(1, sizeof(shared\_memory), IPC\_CREAT | 0666);

if (shm == -1) {

printf("共享内存1创建失败！\n");

exit(0);

}

buf=shmat(shm,NULL,SHM\_R|SHM\_W);

shared=(shared\_memory\*)buf;

int shm2=shmget(2,sizeof(inoutflag),IPC\_CREAT|0666);

if(shm2==-1)

{

printf("共享内存2创建失败！\n");

exit(0);

}

inouttemp=shmat(shm2,NULL,SHM\_R|SHM\_W);

rw=(inoutflag\*)inouttemp;

rw->in=0;

rw->out=0;

rw->length=0;

rw->flag=0;

semid = semget(3,2,IPC\_CREAT|0666); //创建信号灯集

semctl(semid,0,SETVAL,100); //0号灯表示空闲缓冲区个数，当前为number

semctl(semid,1,SETVAL,0); //1号灯表示可读取的缓冲区个数，当前为0

FILE\* wtxt = fopen("/Users/wei/CLionProjects/wei/output.txt","w");

while(1)

{

P(semid,1); //可读的缓冲区减1

printf("write进程读出缓冲区%d\n",rw->out);

if((rw->out+1)%number==rw->in) //这是最后一个缓冲区

{

if(rw->flag==1)

{

fwrite(shared->text[rw->out], sizeof(char),rw->length,wtxt); //打印最后一段的长度

// printf("%s\n",shared->text[rw->out]);

// printf("%d\n",strlen(shared->text[rw->out]));

break;

}

else

{

fwrite(shared->text[rw->out], sizeof(char),size,wtxt); //打印最后一段的长度

}

}

else //否则继续读入

{

fwrite(shared->text[rw->out], sizeof(char),size,wtxt);

rw->out=(rw->out+1)%number;

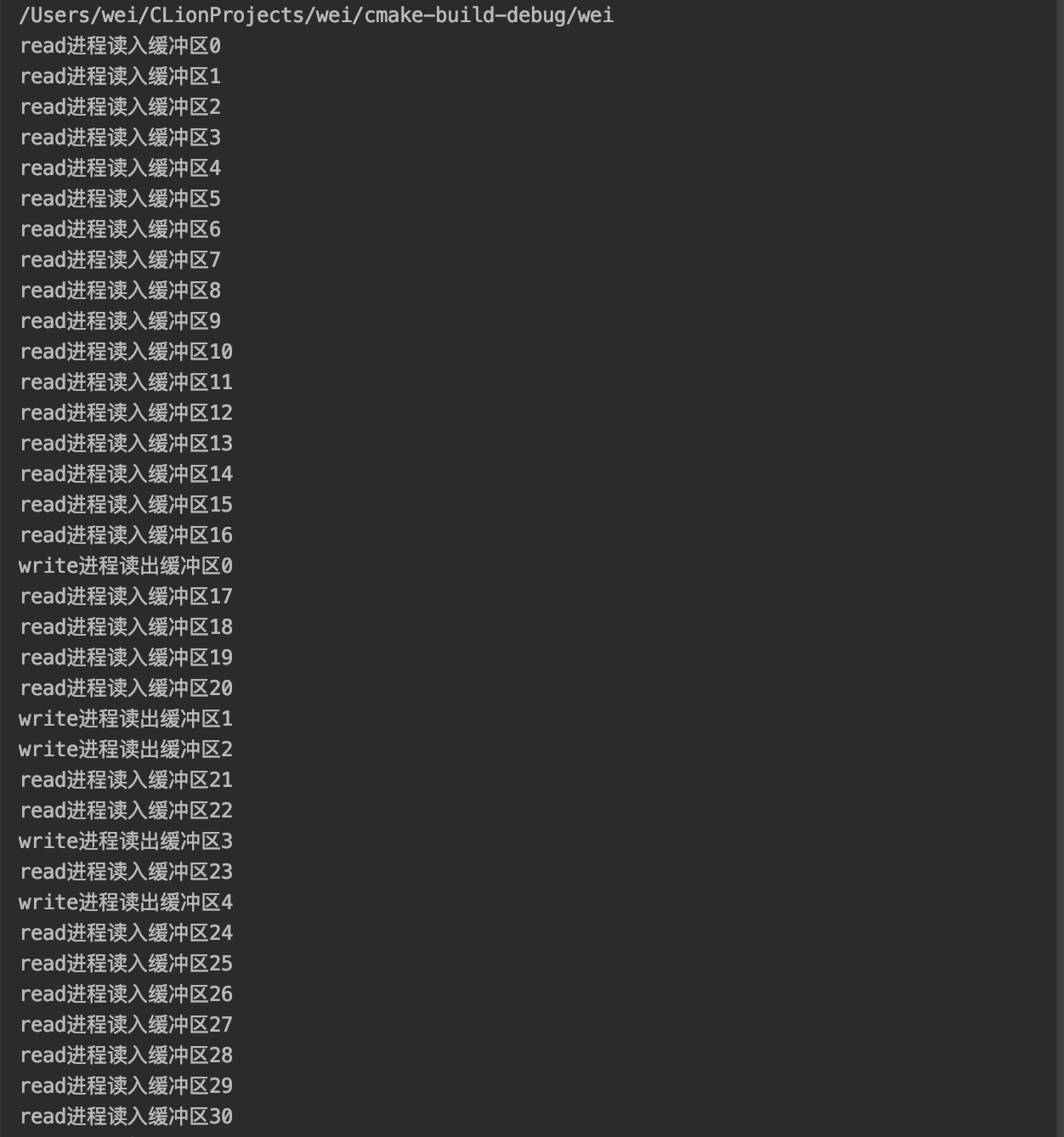
}

V(semid,0);

}

}

1. **实验结果**



输入和输出文件比对结果如下：



两者完全一致。

## 实验心得

这是操作系统的第三次实验。自己的环形缓冲区是由数组实现，由100个共享内存区域构成。并且只要read进程和write进程不在同一个共享内存，它们都可以同时运行。例如read进程正在写入缓冲区50号，那么write进程虽然不可以读出50号区域进程，但除了50号区域内存，其它所有99个共享内存区域都是可以访问的。自己用了两个信号灯实现了上述操作，具体操作可以看查看上面的代码。

另外，自己刚开始写实验3时，read和write两个进程都是由函数实现的，它们都不是实际的程序。但ppt上面要求使用execv（）函数实现，也就是两个进程都应该是可以执行的程序，所以后来自己又重新改成了使用程序实现的版本，最后本实验由主程序，read程序，write程序三个程序同时运行实现。