

**操作系统原理实验报告**

|  |
| --- |
| 姓 名：黄俊淇  学 院：计算机科学与技术 |
| 专 业：计算机科学与技术 |
| 班 级：CS1804 |
| 学 号：U201814604 |
| 指导教师：石柯 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2020年 11月 23日

目 录

[1 实验目的 1](#_Toc27588703)

[2 实验内容 1](#_Toc27588704)

[3 实验设计 1](#_Toc27588705)

[3.1 开发环境 1](#_Toc27588706)

[3.2 实验设计 1](#_Toc27588707)

[4 实验调试 1](#_Toc27588708)

[4.1 实验步骤 1](#_Toc27588709)

[4.2 实验调试及心得 1](#_Toc27588710)

[附录 实验代码 1](#_Toc27588711)

# 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。

# 实验设计

## 开发环境

Linux、CLion

## 实验设计

main1.c：主函数程序，创建信号灯和共享内存，通过fork函数和execv函数创建进程和转到相应进程。如下图代码所示。

图形用户界面, 文本

描述已自动生成

write.c：将源文件内容写入共享内存，通过源文件的字符数量控制循环，将源文件的文本内容写入到共享内存。如下图代码所示。

文本

描述已自动生成

read.c：将共享内存的内容读到目标文件，通过源文件的字符数量控制循环，将共享内存中的文本内容写入到目标文件。如下图代码所示。

文本

描述已自动生成

信号灯：设置两个信号灯，一个控制空缓冲区数量，一个控制满缓冲区数量。

# 实验调试

## 实验步骤

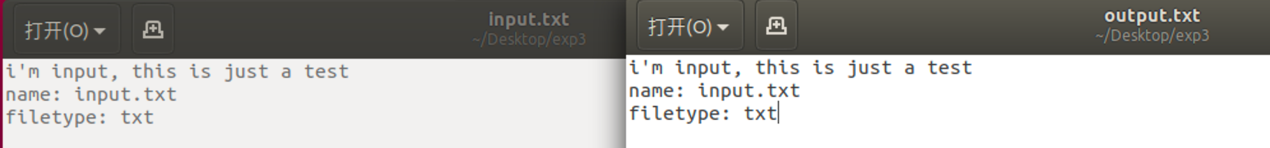
1. 将指定文本内容写入到一个txt文件
2. 创建一个目标txt文件
3. 编译main1.c、read.c、write.c文件
4. 运行./main1

## 实验调试及心得

1、测试结果如下图所示。

文本

描述已自动生成



2、实验结果分析

第一行输出的是共享内存的ID，可以发现每一次都不相同。

通过后面的文本输出可以发现，两个进程是穿插的在运行，甚至后创建的进程会先于先创建的进程运行。

3、心得

相比于实验二采用的全局变量方式，利用共享内存进行通信更具有普适性，关键在于怎么把共享内存ID传给其他进程，我采用的是命令行的方式，以write进程为例，代码如下图所示。

文本

描述已自动生成

在write.c文件中，argv[1]即为共享内存ID。

# 附录 实验代码

main1.c

//

// Created by 黄俊淇的Mac on 2020/11/15.

//

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#define BUFFSIZE 64

pid\_t write\_buf;

pid\_t read\_buf;

union semun {

int val; /\* value for SETVAL \*/

struct semid\_ds \*buf; /\* buffer for IPC\_STAT, IPC\_SET \*/

unsigned short \*array; /\* array for GETALL, SETALL \*/

struct seminfo \*\_\_buf; /\* buffer for IPC\_INFO \*/

};

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);

return;

}

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);

return;

}

struct SHARE{

char buf[BUFFSIZE];

int in,out;

};

int share\_buffer;

int semid;

const char \* source = "/home/parallels/Desktop/exp3/input.txt";

const char \* target = "/home/parallels/Desktop/exp3/output.txt";

struct SHARE \*buffer;

int main(void) {

share\_buffer = shmget(IPC\_PRIVATE, sizeof(struct SHARE) , IPC\_CREAT | 0666);

buffer=(struct SHARE \*)shmat(share\_buffer,NULL,SHM\_R|SHM\_W);

union semun arg;

semid = semget(IPC\_PRIVATE, 2, IPC\_CREAT | 0666); //创建write and read信号

arg.val = BUFFSIZE;

semctl(

semid, 0, SETVAL,

arg); // read进程设置信号灯大小为SIZE,当read到了SIZE大小时会阻塞,等待write使用

arg.val = 0;

semctl(semid, 1, SETVAL, arg); // write

printf("%d\n",share\_buffer);

write\_buf=fork();//将源文件内容写入共享内存

if(write\_buf==0){

char write\_id[20];

char light\_id1[20];

sprintf(write\_id,"%d",share\_buffer);

sprintf(light\_id1,"%d",semid);

printf("%d",share\_buffer);

printf("%s",write\_id);

char \*arg1 = "./write";

char \*arg2 = write\_id;

char \*arg3 = light\_id1;

char \*arg4 = NULL;

char \*arg[] = {arg1, arg2, arg3,arg4};

execv(arg[0], &arg[0]);

}

read\_buf=fork();//将共享内存中的内容输出到目标文件

if(read\_buf==0){

char read\_id[20];

char light\_id2[20];

sprintf(read\_id,"%d",share\_buffer);

sprintf(light\_id2,"%d",semid);

char \*arg1 = "./read";

char \*arg2 = read\_id;

char \*arg3 = light\_id2;

char \*arg4 = NULL;

char \*arg[] = {arg1, arg2, arg3,arg4};

execv(arg[0], &arg[0]);

}

//父进程

waitpid(write\_buf, NULL, 0);

waitpid(read\_buf, NULL, 0);

// remove semaphore

semctl(semid, 1, IPC\_RMID);

// remove share memory

shmctl(share\_buffer, IPC\_RMID, NULL);

exit(0);

}

write.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#define BUFFSIZE 64

union semun {

int val; /\* value for SETVAL \*/

struct semid\_ds \*buf; /\* buffer for IPC\_STAT, IPC\_SET \*/

unsigned short \*array; /\* array for GETALL, SETALL \*/

struct seminfo \*\_\_buf; /\* buffer for IPC\_INFO \*/

};

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);

return;

}

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);

return;

}

struct SHARE{

char buf[BUFFSIZE];

int in,out;

};

int share\_buffer;

int semid;

const char \* source = "/home/parallels/Desktop/exp3/input.txt";

const char \* target = "/home/parallels/Desktop/exp3/output.txt";

struct SHARE \*buffer;

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

share\_buffer=atoi(argv[1]);

semid=atoi(argv[2]);

buffer=(struct SHARE \*)shmat(share\_buffer,NULL,SHM\_R|SHM\_W);

FILE \*fp=fopen(source,"r");

int count;

printf("将源文件内容写入共享内存\n");

fseek(fp,0,SEEK\_END);

long length=ftell(fp);

fseek(fp,0,SEEK\_SET);

buffer->in=0;

for(count=0;count<length;count++){

P(semid,0);

buffer->buf[buffer->in]=fgetc(fp);

printf("%c",buffer->buf[buffer->in]);

buffer->in++;

buffer->in%=BUFFSIZE;

V(semid,1);

}

fclose(fp);

shmdt(buffer);

exit(0);

}

read.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#define BUFFSIZE 64

union semun {

int val; /\* value for SETVAL \*/

struct semid\_ds \*buf; /\* buffer for IPC\_STAT, IPC\_SET \*/

unsigned short \*array; /\* array for GETALL, SETALL \*/

struct seminfo \*\_\_buf; /\* buffer for IPC\_INFO \*/

};

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);

return;

}

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);

return;

}

struct SHARE{

char buf[BUFFSIZE];

int in,out;

};

int share\_buffer;

int semid;

const char \* source = "/home/parallels/Desktop/exp3/input.txt";

const char \* target = "/home/parallels/Desktop/exp3/output.txt";

struct SHARE \*buffer;

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

share\_buffer=atoi(argv[1]);

semid=atoi(argv[2]);

buffer=(struct SHARE \*)shmat(share\_buffer,NULL,SHM\_R|SHM\_W);

int count;

printf("将共享内存中的内容输出到目标文件\n");

//P(semid,1);

FILE \*fp=fopen(source,"r");

fseek(fp,0,SEEK\_END);

long length=ftell(fp);

fclose(fp);

FILE \*fp1;

fp1=fopen(target,"w");

buffer->out=0;

for(count=0;count<length;count++){

P(semid,1);

printf("%c",buffer->buf[buffer->out]);

fputc(buffer->buf[buffer->out],fp1);

buffer->out++;

buffer->out=buffer->out%BUFFSIZE;

V(semid,0);

}

fclose(fp1);

shmdt(buffer);

exit(0);

}