

華中科技大學

# 课程实验报告

课程名称：操作系统原理

专业班级：CS1902

学号：I201920024

姓名：木林

指导教师：胡贯荣

报告日期：2022年1月1日

计算机科学与技术学院

# 实验三：共享内存与进程同步

## 一、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。
- 2、掌握Linux下进程同步与通信的主要机制。

## 二、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。

·主进程:主进程用于初始化环形缓冲区、启动两个子进程并等待两个子进程的结束。

·读进程:读进程用于从文件中读取数据并将数据写入环形缓冲区。

·写进程:写进程用于从环形缓冲区中读取数据并写入新文件中。

线程之间的同步通过三个信号量进行，分别为 sem0, sem1、sem2 用于标识读进程读入的数据量, sem1 用于标识环形缓冲中剩余的空间, sem2 用于进程间的互斥，即同时只允许一个进程访问环形缓冲区.如下图1.1

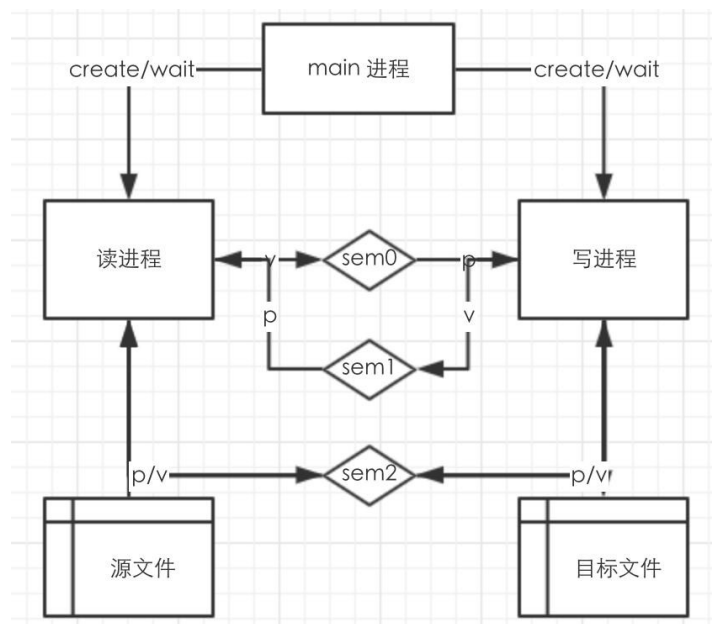


图1.1系统操作流程图

## 2、运行环境



图2.1本机系统

·笔记本电脑系统 macOS Big Sur.

·UTM GEMU 6.1 ARM 虚拟机 ubuntu-20.02.3

·gcc version 12.8 ubuntu 1.1

## 3、源程序

```
#include <sys/types.h>
#include <sys/sem.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <stdbool.h>

#define BUFFSIZE 100

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 = SEM_UNDO;
    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 = SEM_UNDO;
    semop(semid, &sem, 1);
    return ;
}

int main(void) {
    // 信号灯集
    int semid;
    // 创建共享分区, 大小为buff
    int share_buffer = shmget(IPC_PRIVATE, sizeof(unsigned char) * BUFFSIZE,
IPC_CREAT | 0666);
    // 先暂时这么写
    const char * source = "input.txt";
    const char * target = "output.txt";

    if ((semid = semget(IPC_PRIVATE, 0, 0) == -1)) {
        // 如果信号灯集不存在的话, 就创建三个信号灯, 第三个信号灯用于判断是否
        完成
        if ((semid = semget(IPC_PRIVATE, 2, IPC_CREAT | 0666)) != -1) {
            // 如果创建成功
            union semun w_arg; // 用于写进程的参数
            union semun r_arg; // 用于读进程的参数
            // union semun lock;
            w_arg.val = BUFFSIZE; // 写进程的初始值为BUFFSIZE
            r_arg.val = 0; // 读进程的初始值为0
            // lock.val = 1;
            // 第一个信号灯给写进程, 第二个给读进程
            if (semctl(semid, 0, SETVAL, w_arg) == -1 ||
                semctl(semid, 1, SETVAL, r_arg) == -1) {
                perror("IPC error 1: semctl");
                exit(1);
            }
        }
        int write_buff = fork();
        if (write_buff == 0) {
            // unsigned long * file_len_tmp = (unsigned long
            *)shmat(file_len_share, NULL, SHM_W);
            unsigned long counter;
            unsigned char c;
            unsigned long file_len;
            // int * in_index = (int *)shmat(in_index, NULL, SHM_W);
            int in_index = 0;
            FILE * source_file = fopen(source, "rb");

            unsigned char * buffer = (unsigned char *)shmat(share_buffer, NULL,
0);

            // printf("进入写入方\n");
            fseek(source_file, 0, SEEK_END);
            // printf("获取文件长度\n");
            // *file_len_tmp = ftell(source_file); // 获取文件长度
            file_len = ftell(source_file);
            // printf("写入方得到的文件长度为%d", file_len);

            fseek(source_file, 0, SEEK_SET);

```

```

        for (counter = 0; counter < file_len; counter++) {
            c = fgetc(source_file);
            P(semid, 0);
            *(buffer + (in_index % BUFFSIZE)) = c;
            in_index++;
            in_index %= BUFFSIZE;
            V(semid, 1);
            // printf("从文件读出%c, counter = %ld, file_len = %ld\n", c, counter,
file_len);
        }
        printf("写入进程结束\n");

    } else {
        int read_buff = fork();
        if (read_buff == 0) {
            // 读进程
            // printf("进入读进程\n");
            unsigned char c;
            FILE * target_file = fopen(target, "wb");
            FILE * source_file = fopen(source, "rb");
            fseek(target_file, 0, SEEK_SET);
            int out_index = 0;
            unsigned char * buffer = (unsigned char *)shmat(share_buffer,
NULL, 0);
            // printf("读进程尝试获取长度\n");
            fseek(source_file, 0, SEEK_END);
            // printf("读进程获取文件长度\n");
            unsigned long file_len = ftell(source_file);
            // printf("获取长度成功\n");
            fclose(source_file);
            // printf("读取方得到的文件长度为%ld", file_len);
            unsigned long counter;
            for (counter = 0; counter < file_len; counter++) {
                P(semid, 1);
                c = *(buffer + out_index % BUFFSIZE);
                out_index++;
                out_index %= BUFFSIZE;
                fputc(c, target_file);
                V(semid, 0);
                // printf("写入文件%c, counter = %ld length = %ld\n", c, counter,
file_len);
            }
            printf("读出进程结束\n");
        } else {
            // 父进程
            waitpid(read_buff, NULL, 0);
            waitpid(write_buff, NULL, 0);
            // 销毁信号灯
            if (semctl(semid, 0, IPC_RMID) == -1) {
                perror("Destroy Semaphore Failed!\n");
            }
            // 销毁共享内存
            if (shmctl(share_buffer, IPC_RMID, NULL) == -1) {
                perror("Destroy share memory failed!\n");
            }
        }
    }
}

} else {

```

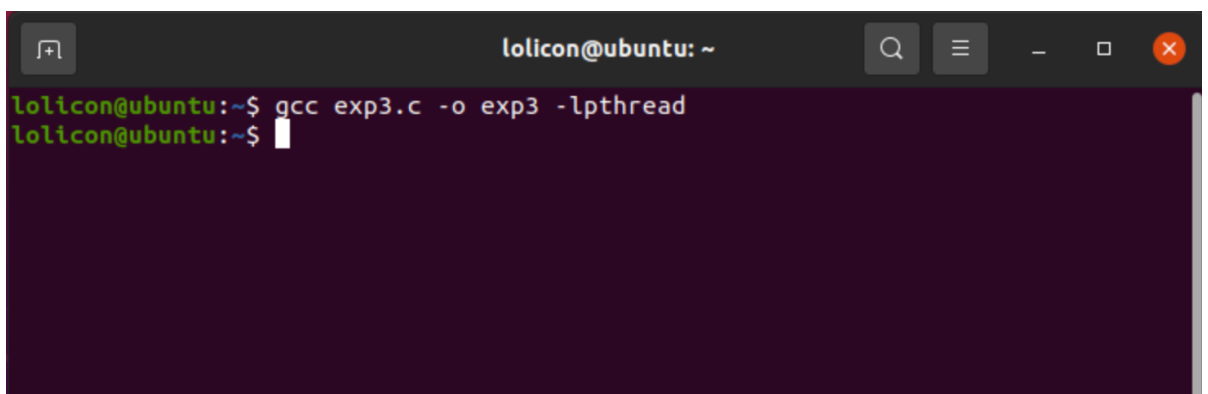
```

        perror("IPC error2: semget");
        exit(1);
    }
} else {
    perror("Semaphore is already created!\n");
    exit(1);
}
return 0;
}

```

## 4、实验结果

·通过 terminal 进行编译.命令为 `gcc exp3.c -o exp3 -lpthread`, 其中exp3为本实验文件的名字, 得到可执行文件.如图4.1下



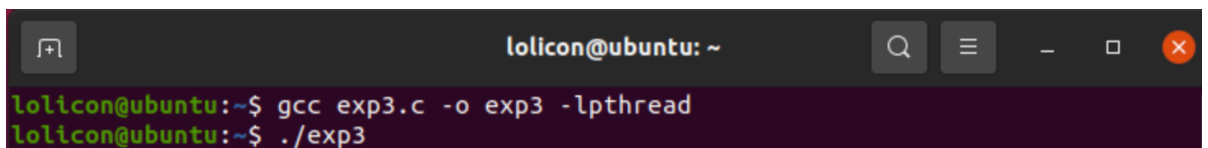
```

lolicon@ubuntu: ~
lolicon@ubuntu:~$ gcc exp3.c -o exp3 -lpthread
lolicon@ubuntu:~$

```

图4.1操作指令1

·然后直接使用 `./exp3` 进行测试.如图4.2下



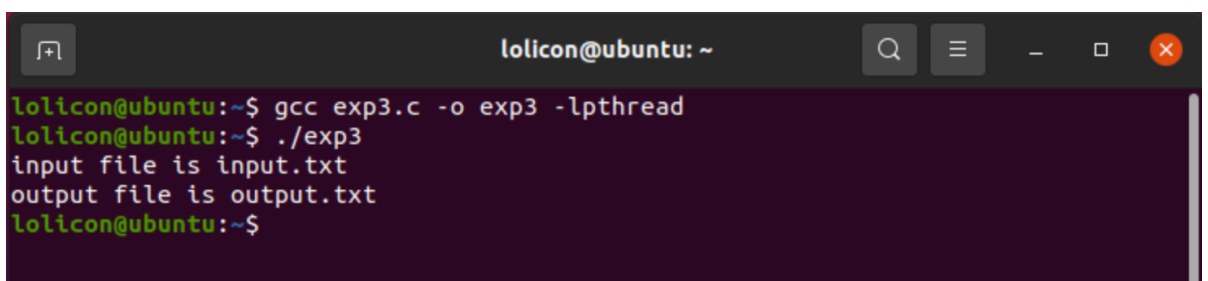
```

lolicon@ubuntu: ~
lolicon@ubuntu:~$ gcc exp3.c -o exp3 -lpthread
lolicon@ubuntu:~$ ./exp3

```

图4.2操作指令2

·如果操作成功, 显示如图4.3下



```

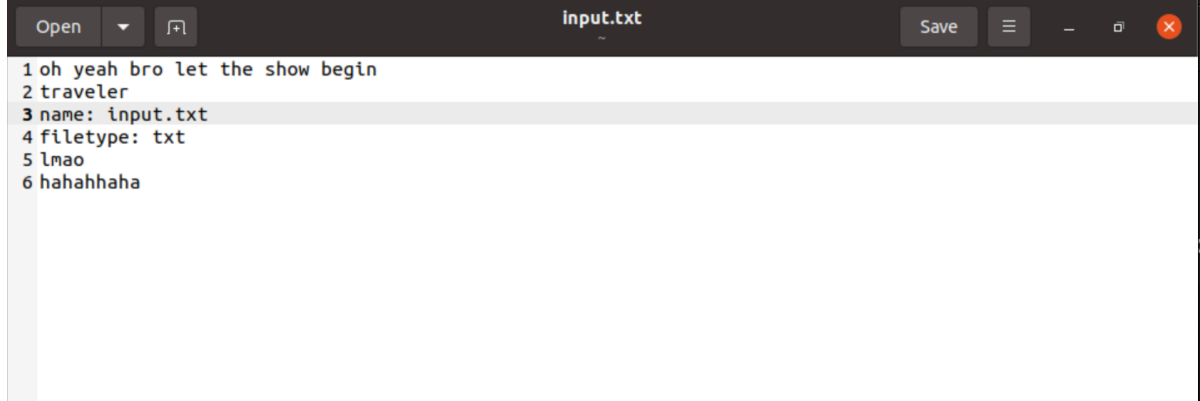
lolicon@ubuntu:~$ gcc exp3.c -o exp3 -lpthread
lolicon@ubuntu:~$ ./exp3
input file is input.txt
output file is output.txt
lolicon@ubuntu:~$

```

图4.3成功显示

然后在 ubuntu 中导航 File->Home 然后查看 output.txt 以检查结果。请注意, 在执行上述所有步骤之前需要 input.txt (我们要复制其内容的文件)

测试:



```
1 oh yeah bro let the show begin
2 traveler
3 name: input.txt
4 filetype: txt
5 lmao
6 hahahahaha
```

-input.txt 我们想要复制其内容的文件。如图4.4下



```
1 oh yeah bro let the show begin
2 traveler
3 name: input.txt
4 filetype: txt
5 lmao
6 hahahahaha
```

图4.4 input.txt

-操作之后得到结果文件为output.txt。如图4.5下

图4.5 output.txt

### 三、实验心得

这是我第一次从操作系统级别编写代码。由于之前没有真正处理过信号量和共享内存的实际操作，我花了很多时间看资料和学习实际程序。当两个进程使用页表将一个虚拟地址映射到一个物理地址时，在其中有一个共享内存空间两个进程可能同时看到的物理地址。为了完成进程间通信，当一个进程进行写操作时，另一个进程读取该操作。为了保证进程在写入时不被读取，我们利用信号量来提供同步和互斥。在本实验中，我使用了三个信号量来实现进程同步和互斥。信号量值为0。

为了表示接收到的数据量，使用信号量1，使用信号量2进行锁定，因此信号量0、1、2的初始值分别赋值为0，环形缓冲区节点数为1。最后，尝试启动读取和写入过程，然后等待这两个过程完成。两个程序完成后，释放所有请求的资源。如果在整个主线程执行过程中打开文件、生成缓冲区、创建信号量或启动子进程中的任何一个阶段未能成功运行，立即释放分配的资源并退出程序。

以上是本次实验的重点。

在编写过程中，我主要是从手册中学习如何使用 shm 和 sem。

感谢各位导师的指导和帮助。