华中科技大学

操作系统原理实验报告

姓　　名：　木林

学　　院：　计算机学院

专　　业：　计算机电脑

班　　级：　CS1902

学　　号：　I201920024

指导教师：　胡贯荣

分数	
教师签名	

年 2022 月 1 日 1

目　录

实验三 共享内存与进程同步

1. 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。

2. 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。

3. 实验设计

3.1. 开发环境

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

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**3.2**

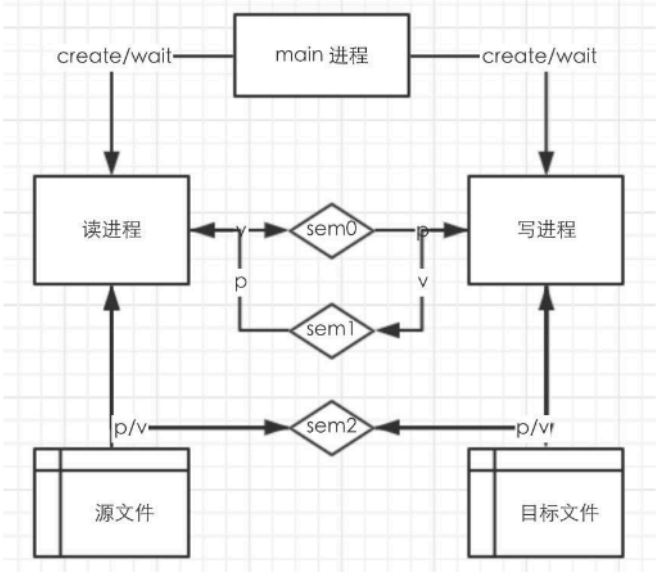
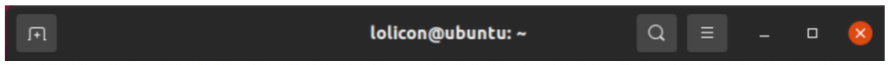


图3.2操作流程图

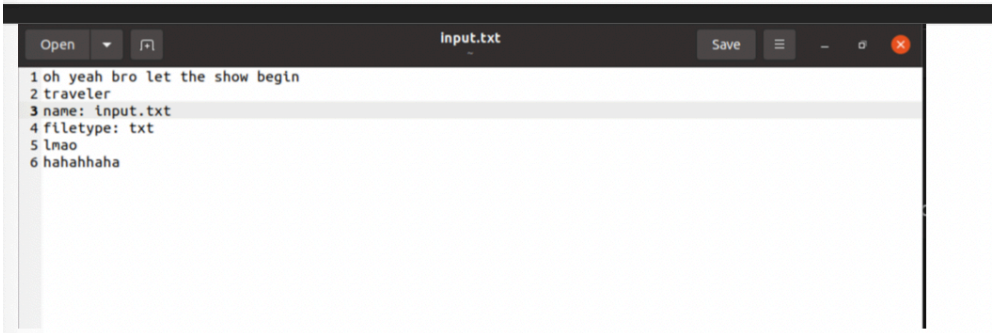
4. 实验调试

4.1. 实验步骤

·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



测试：



-input.txt 我们想要复制其内容的文件。如图4.4下

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

4.2. 实验调试及心得

附录 实验代码

```
#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%ld", 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;
}
```