



# 华中科技大学

## 操作系统原理实验报告

姓 名： 刘日星  
学 院： 计算机科学与技术  
专 业： 计算机科学与金融  
班 级： CS1804（交换）  
学 号： X2020I1007  
指导教师： 胡贯荣

分数	
教师签名	

2021 年 12 月 23 日

## 目 录

1.1 实验目的 .....	1
1.2 实验内容 .....	1
1.3 实验设计 .....	1
1.3.1 开发环境 .....	1
1.3.2 实验设计 .....	2
1.4 实验调试 .....	3
1.4.1 实验步骤 .....	3
1.4.2 实验调试及心得 .....	5
附录 实验代码 .....	5

# 实验三、共享内存与进程同步

## 1.1 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 进一步掌握 Linux 下进程控制及程序加载;
- 4 进一步掌握 Linux 下进程同步与通信的主要机制。

正文统一采用小四号宋体/Times New Roman 和 1.25 倍行距。

## 1.2 实验内容

- ◆ 利用 1 个或多个共享内存（有限空间）构造环形缓冲;
- ◆ 利用环形缓冲，设置信号灯，实现两个进程的誊抄;

GET→环形缓冲→PUT

注：GET 和 PUT 必须是独立的可执行程序

- ◆ 誊抄将源文件正确复制到目标文件，源文件可以是小文件、二进制文件、大文件及正常大小文件;
- ◆ 选择考虑三个进程的誊抄（利用双环形缓冲），不作要求。

GET→环形缓冲 1→COPY→环形缓冲 2→PUT

## 1.3 实验设计

### 1.3.1 开发环境

系统：Windows 10 内装虚拟机 ubuntu 20.04.2 LTS 64 位  
内存（虚拟机）：8G

处理器：Intel®Core™ i7-7700HQ CPU @ 2.80GHz 2.81 GHz

硬盘(虚拟机)：60G

显卡：Intel® HD Graphics 630; NVIDIA GTX 1050 4G

编译器版本：GCC 9.3.0

调试器版本：GNU 9.2

### 1.3.2 实验设计

本次实验一共建立了三个环形缓冲区在 main 文件里，分别提供给 GET 和 PUT 进行调用。

#### main 主程序：

第一步：通过调用函数 `int shmget(key, size, flag)` 格式创建 3 个大小相等 KEY 的共享缓冲区（共享存储区），分别为 `shmid1`、`shmid2` 和 `shmid3`，每个共享缓冲区大小统一为 100 并且对其进行初始化。

第二步：继续调用函数 `int shmget(key_t key, int nsems, int semflg)` 格式创建 3 个 KEY 信号灯，分别为 `arg1`、`arg2` 和 `arg3`，同时为三个信号灯进行赋值，而三个信号灯的编号分别为 0、1、2。

第三步：在主程序内创建两个子进程 GET 和 PUT，分别命名为 `pid_g` 和 `pid_p`。若子进程创建成功则使用 `execv` 分别调用运行编译完成的 GET 文件内的程序和 PUT 文件内的程序。

第四步：等待 GET 子进程和 PUT 子进程运行完毕后，调用 `shmctl(shmid, IPC_RMID, 0)` 格式删除共享缓冲区 `shmid1`、`shmid2` 和 `shmid3` 并释放资源。然后再调用相同函数格式对三个信号灯进行删除和释放资源。结束。

#### GET 操作：

第一步：通过调用 `int shmget(key, size, flag)` 格式对已创建好的 `shmid1`、`shmid2` 和 `shmid3` 三个 KEY 的共享缓冲区进行获取。然后使用函数 `int shmget(key_t key, int nsems, int semflg)` 格式获取 KEY 的三个信号灯。

第二步：调用系统函数 `viraddr=shmat(shmid, addr, flag)` 格式建立数组形式环形缓冲区 `buf[0]`、`buf[1]` 和 `buf[2]` 并且分别获取 KEY 的共享缓冲区 `shmid1`、`shmid2` 和 `shmid3` 的首地址。

第三步：以只读 “r” 形式打开源文件 “input.txt”。

第四步：使用 `while` 循环和信号灯 P 操作对 `input.txt` 文件内的数据进行循环读取。调用 `fgets()` 函数进行判断读取文件数据是否成功。若成功则将文件内的数据读入到有序的三个环形缓冲区内，每当存满一个环形缓冲区时则调用信号灯 V 操作唤醒环形缓冲区的指针指向下一个环形缓冲区（环形缓冲区的指针移动），然后继续循环读入数据。

第五步：当 `input.txt` 文件内的数据被读完后调用 V 操作，`fclose()` 函数关闭对文件的操作，最后调用一个信号灯 V 操作唤醒下一个进程和调用 `exit(0)` 完成对 GET 的操作。

## PUT 操作:

第一步:通过调用 `int shmget(key, size, flag)` 格式对已创建好的 `shmid1`、`shmid2` 和 `shmid3` 三个 KEY 的共享缓冲区进行获取。然后使用函数 `int shmget(key_t key, int nsems, int semflg)` 格式获取 KEY 的三个信号灯。

第二步:调用系统函数 `viraddr=shmat(shmid, addr, flag)` 格式建立数组形式环形缓冲区 `buf[0]`、`buf[1]` 和 `buf[2]` 并且分别获取 KEY 的共享缓冲区 `shmid1`、`shmid2` 和 `shmid3` 的首地址。

第三步:以只写“w”的形式创建一个名为“output.txt”的目标文件。

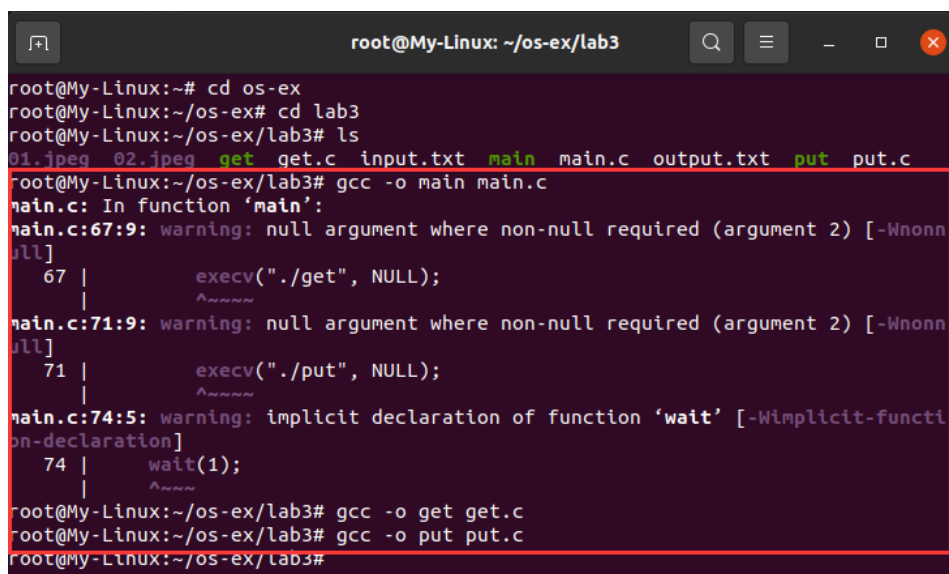
第四步:使用 `while` 循环和信号灯 P 操作对环形缓冲区内的数据进行循环获取,每当获取完一个环形缓冲区内的数据后调用信号灯 V 操作和调用 `fputs()` 函数对从环形缓冲区内获取出来的数据进行写入 `output.txt` 文件内,然后移动环形缓冲区指针指向下一个环形缓冲区进行读取数据并写入文件。

第五步:调用函数 `stccmp()` 进行判断每个环形缓冲区是否为空,若所有环形缓冲区都为空,则为环形缓冲区内的数据获取完成以及数据写入 `output.txt` 文件内完成。调用 `fclose()` 函数关闭对文件的操作,最后调用一个信号灯 V 操作唤醒下一个进程和调用 `exit(0)` 完成对 PUT 的操作。

## 1.4 实验调试

### 1.4.1 实验步骤

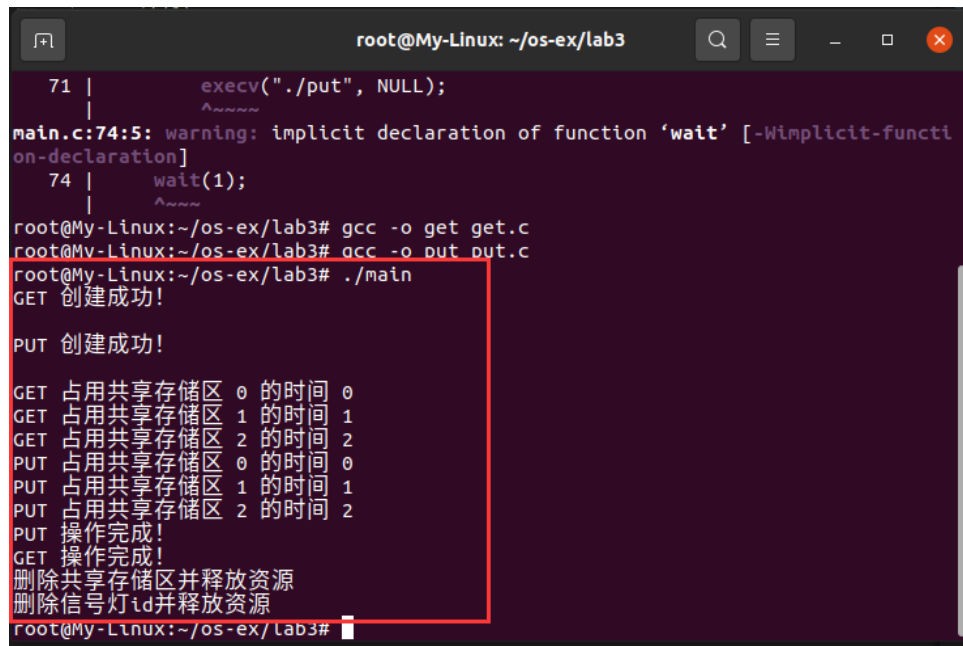
第一步:对已写好的 `main.c` 文件、`get.c` 文件和 `put.c` 文件进行 gcc 编译,如图 1-4-1:



```
root@My-Linux: ~/os-ex/lab3
root@My-Linux:~# cd os-ex
root@My-Linux:~/os-ex# cd lab3
root@My-Linux:~/os-ex/lab3# ls
01.jpeg 02.jpeg get get.c input.txt main main.c output.txt put put.c
root@My-Linux:~/os-ex/lab3# gcc -o main main.c
main.c: In function 'main':
main.c:67:9: warning: null argument where non-null required (argument 2) [-Wnonnull]
    67 |         execv("./get", NULL);
        |         ~~~~~
main.c:71:9: warning: null argument where non-null required (argument 2) [-Wnonnull]
    71 |         execv("./put", NULL);
        |         ~~~~~
main.c:74:5: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
    74 |         wait(1);
        |         ~~~~
root@My-Linux:~/os-ex/lab3# gcc -o get get.c
root@My-Linux:~/os-ex/lab3# gcc -o put put.c
root@My-Linux:~/os-ex/lab3#
```

图 1-4-1

第二步：运行编译好的 main 文件将源文件 input.txt 内的数据誊抄到目标文件 output.txt 内，如图 1-4-2：



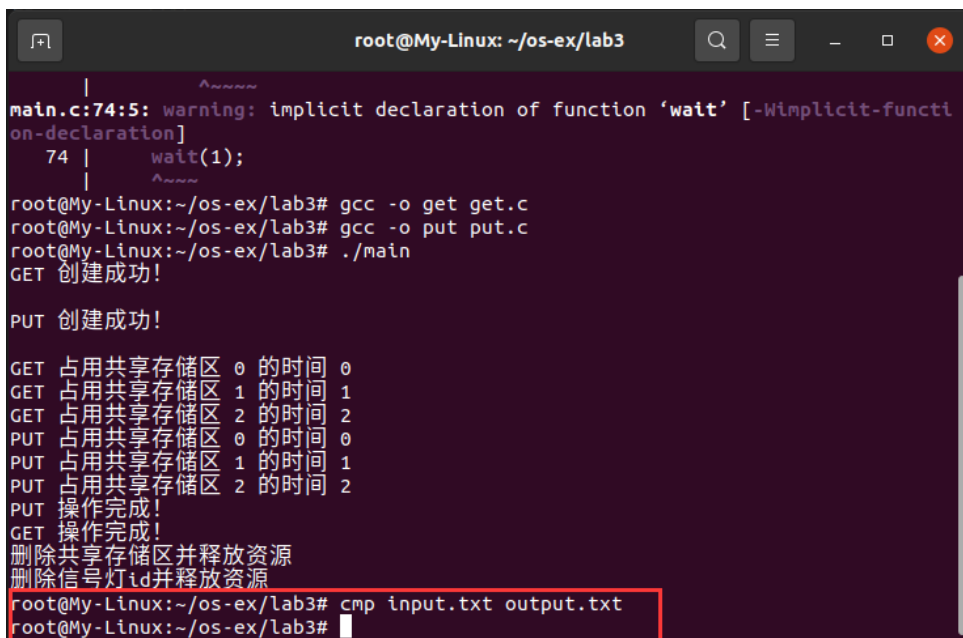
```
71 |         execv("./put", NULL);
    |
main.c:74:5: warning: implicit declaration of function 'wait' [-Wimplicit-functi
on-declaration]
74 |         wait(1);
    |
root@My-Linux:~/os-ex/lab3# gcc -o get get.c
root@My-Linux:~/os-ex/lab3# gcc -o put put.c
root@My-Linux:~/os-ex/lab3# ./main
GET 创建成功!

PUT 创建成功!

GET 占用共享存储区 0 的时间 0
GET 占用共享存储区 1 的时间 1
GET 占用共享存储区 2 的时间 2
PUT 占用共享存储区 0 的时间 0
PUT 占用共享存储区 1 的时间 1
PUT 占用共享存储区 2 的时间 2
PUT 操作完成!
GET 操作完成!
删除共享存储区并释放资源
删除信号灯id并释放资源
root@My-Linux:~/os-ex/lab3#
```

图 1-4-2

第三步：选择 txt 类型文件进行测试，使用 cmp 命令比较 input.txt 和 output.txt 的差异，如图 1-4-3：



```
main.c:74:5: warning: implicit declaration of function 'wait' [-Wimplicit-functi
on-declaration]
74 |         wait(1);
    |
root@My-Linux:~/os-ex/lab3# gcc -o get get.c
root@My-Linux:~/os-ex/lab3# gcc -o put put.c
root@My-Linux:~/os-ex/lab3# ./main
GET 创建成功!

PUT 创建成功!

GET 占用共享存储区 0 的时间 0
GET 占用共享存储区 1 的时间 1
GET 占用共享存储区 2 的时间 2
PUT 占用共享存储区 0 的时间 0
PUT 占用共享存储区 1 的时间 1
PUT 占用共享存储区 2 的时间 2
PUT 操作完成!
GET 操作完成!
删除共享存储区并释放资源
删除信号灯id并释放资源
root@My-Linux:~/os-ex/lab3# cmp input.txt output.txt
root@My-Linux:~/os-ex/lab3#
```

图 1-4-3

## 1.4.2 实验调试及心得

本次实验总体来说有一定的难度，对于我来说是一份锻炼。通过此次实验，让我对操作系统的进程间同步和互斥访问共享存储区有了更深的了解。不过在完成实验的过程时，遇到了一些问题，在仔细检查代码和网上查找问题的解决方法后，逐步修复问题。例如，在写主程序 `main` 的时候，总会发现漏掉一些头文件的调用，致使后面使用的一些函数会报错显示不允许使用，所以就边写程序边查找添加所需头文件；以及在 `GET` 内将数据循环读入一个环形缓冲区并且装满一个缓冲区后，程序没有自动唤醒下一个，所以就查问题所在直到发现需要在循环判断缓冲区是否已满里添加一个 `V` 操作唤醒下一个环形缓冲区。综合来说，这次实验让我学到了不少知识，对进程间同步和互斥访问共享存储区的实现有了足够的实践和掌握。同时了解了计算机操作系统中文件复制粘贴操作实现的原理。

## 附录 实验代码

**main 主程序：**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/sem.h>

#define SHMKEY1 1111
#define SHMKEY2 2222
#define SHMKEY3 3333

int semid;           //定义一个信号灯 id
int shmid1, shmid2, shmid3; //定义 3 个共享存储区
char *addr1, *addr2, *addr3; //定义 3 个共享缓冲区地址
```

```

int pid_p, pid_g;

union semun{
    int val;
    struct semid_ds* buf;
    unsigned short* array;
};

// P 操作接口
void P(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;    //信号灯值-1 (s 值-1)
    sem.sem_flg = 0;    //操作标记: 0 或 IPC_NOWAIT 等
    semop(semid, &sem, 1); //1: 表示执行命令的个数
    return;
}

// V 操作接口
void V(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;    //信号灯值+1
    sem.sem_flg = 0;    //操作标记: 0
    semop(semid, &sem, 1);
    return;
}

int main(){
    // 创建 3 个共享存储区
    shmid1 = shmget(SHMKEY1, 100*sizeof(char), 0666|IPC_CREAT);
    shmid2 = shmget(SHMKEY2, 100*sizeof(char), 0666|IPC_CREAT);
    shmid3 = shmget(SHMKEY3, 100*sizeof(char), 0666|IPC_CREAT);

```



```

    semid = semget((key_t)13, 3, IPC_CREAT|0666);    //key 信号灯创建,
key_t 是共享存储区名字
    union semun arg1, arg2, arg3;
    arg1.val = 3;
    semctl(semid, 0, SETVAL, arg1);    /*对信号灯的控制操作*/
    arg2.val = 0;
    semctl(semid, 1, SETVAL, arg2);
    arg3.val = 0;
    semctl(semid, 2, SETVAL, arg3);

// 进程控制，系统调用 GET 和 PUT
if((pid_g = fork()) == 0){
    puts("GET 创建成功! \n");
    execv("./get", NULL);
}
else if((pid_p = fork()) == 0){
    puts("PUT 创建成功! \n");
    execv("./put", NULL);
}

wait(1);
wait(1);

printf("删除共享存储区并释放资源\n");
shmctl(shmid1, IPC_RMID, 0);    /*撤销共享存储区，释放资源*/
shmctl(shmid2, IPC_RMID, 0);
shmctl(shmid3, IPC_RMID, 0);

printf("删除信号灯 id 并释放资源\n");
semctl(semid, IPC_RMID, 0);    /*删除信号灯 id，释放资源*/

return 0;
}

```

## GET 程序:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stddef.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/sem.h>
```

```
#define SHMKEY1 1111
#define SHMKEY2 2222
#define SHMKEY3 3333
```

```
int semid;                //定义一个信号灯 id
int shmid1, shmid2, shmid3; //定义 3 个共享存储区
char *buf[3];             //设立缓冲区个数
```

```
union semun{
    int val;
    struct semid_ds* buf;
    unsigned short* array;
};
```

// P 操作接口

```
void P(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;    //信号灯值-1 (s 值-1)
    sem.sem_flg = 0;    //操作标记: 0 或 IPC_NOWAIT 等
    semop(semid, &sem, 1); //1: 表示执行命令的个数
    return;
}
```

```

// V 操作接口
void V(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;    //信号灯值+1
    sem.sem_flg = 0;    //操作标记: 0
    semop(semid, &sem, 1);
    return;
}

int main(){
    // 获取 key 的 3 个共享存储区
    shmid1 = shmget(SHMKEY1, 100*sizeof(char), 0666);
    shmid2 = shmget(SHMKEY2, 100*sizeof(char), 0666);
    shmid3 = shmget(SHMKEY3, 100*sizeof(char), 0666);

    // 建立数组形式环形缓冲区
    buf[0] = (char*)shmat(shmid1, 0, 0);    /*获取 shmid1 的首地址*/
    buf[1] = (char*)shmat(shmid2, 0, 0);    /*获取 shmid2 的首地址*/
    buf[2] = (char*)shmat(shmid3, 0, 0);    /*获取 shmid3 的首地址*/

    semid = semget((key_t)13, 3, IPC_CREAT|0666);    // 获取 Key 的信号灯

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

    int i = 0;
    int j = 0;
    char c[100] = "EOF";
    while(1){
        P(semid, 0);    // 信号灯 P 操作
        if((fgets(buf[i], 100, fileprocess)) == NULL){    //判断读取文件内容
            是否完成
            strcpy(buf[i], c);
        }
    }
}

```

```

        V(semid, 1);
        fclose(fileprocess);
        break;
    }
    V(semid, 1);        //信号灯 V 操作
    printf("GET 占用共享存储区 %d 的时间 %d\n", j++, i);
    i = (i + 1) % 3;    //移动环形缓冲区指针
}
P(semid, 2);
printf("GET 操作完成! \n");
exit(0);
}

```

#### **PUT 程序:**

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stddef.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/sem.h>

#define SHMKEY1 1111
#define SHMKEY2 2222
#define SHMKEY3 3333

int semid;                //定义一个信号灯 id
int shmid1, shmid2, shmid3; //定义 3 个共享存储区
char *buf[3];            //设立缓冲区个数

union semun{

```

```

    int val;
    struct semid_ds* buf;
    unsigned short* array;
};

// P 操作接口
void P(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = -1;    //信号灯值-1 (s 值-1)
    sem.sem_flg = 0;    //操作标记: 0 或 IPC_NOWAIT 等
    semop(semid, &sem, 1); //1: 表示执行命令的个数
    return;
}

// V 操作接口
void V(int semid, int index){
    struct sembuf sem;
    sem.sem_num = index;
    sem.sem_op = 1;    //信号灯值+1
    sem.sem_flg = 0;    //操作标记: 0
    semop(semid, &sem, 1);
    return;
}

int main(){
    // 获取 key 的 3 个共享存储区
    shmid1 = shmget(SHMKEY1, 100*sizeof(char), 0666);
    shmid2 = shmget(SHMKEY2, 100*sizeof(char), 0666);
    shmid3 = shmget(SHMKEY3, 100*sizeof(char), 0666);

    // 建立数组形式环形缓冲区
    buf[0] = (char*)shmat(shmid1, 0, 0);    /*获取 shmid1 的首地址*/
    buf[1] = (char*)shmat(shmid2, 0, 0);    /*获取 shmid2 的首地址*/
    buf[2] = (char*)shmat(shmid3, 0, 0);    /*获取 shmid3 的首地址*/
}

```

```

semid = semget((key_t)13, 3, IPC_CREAT|0666);    // 获取 Key 的信号灯

FILE *fileprocess;

if((fileprocess = fopen("output.txt", "w")) == NULL){    /* 创建一个名为
output.txt 的目标写入文件*/
    printf("文件打开出错!\n");
    exit(0);
}

int i = 0;
int j = 0;
char c[100] = "EOF";
while(1){
    P(semid, 1);
    if(strcmp(buf[i], c) == 0){
        fclose(fileprocess);
        break;
    }
    V(semid, 0);    //信号灯 V 操作
    fputs(buf[i], fileprocess);    /*从共享缓冲区内提取数据并写入文件*/
    printf("PUT 占用共享存储区 %d 的时间 %d\n", j++, i);
    i = (i + 1) % 3;    //移动环形缓冲区指针
}
V(semid, 2);
printf("PUT 操作完成! \n");
exit(0);
}

```