Semaphore 使用实验

一、题目

使用 semaphore，并利用该程序生成 2 个进程（注意：非线程），这两个进程写同一个文件，要求：

互斥写，即只有一个进程写完后，才能让另一个进程写；

一个进程写入内容：“自己学号的后 3 位 PROC1 MYFILE1”；

另一个进程写入内容：“自己学号的后 3 位 PROC2 MYFILE2”），将该程序的 semaphore 替换成使用 strict alternation 算法的忙等待互斥锁完成。

回答问题：

i. 自己程序中关键句的含义

ii. 请用实际操作证明当进程 A 占用 semaphore 后，进程 B 想要占用 semaphore 时，进程 B 进入睡眠。

iii.移植 Modern Operating System 一书中的 strict alternation 算法时，该算法中的 turn 变量访问时是否需要加锁，以避免读写冲突？

二、编写程序

1.sem1.c（信号量初始值为 1）

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main(){
    int semid;
    union semun{
        int val;
        struct semid_ds *buf;
        unsigned short *array;
        struct seminfo *_buf;
    } semval;

    semval.val = 1;
    semid = semget(0x123, 1, IPC_CREAT | IPC_EXCL | 0600);
    semctl(semid, 0, SETVAL, semval);
}
```

2. semaphore1.c

```c
#include <stdio.h>
#include <sys/types.h>
```

```c
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>

int main(){
    int flag = semget(0x123, 1, 0);

    while(1) {
        while(semctl(flag, 0, GETVAL) != 1); // 等待flag为1

        FILE *fp = fopen("out.txt", "a");
        fprintf(fp, "894 PROC1 MYFILE1\n");
        fclose(fp);
        printf("flag = %d\n", semctl(flag, 0, GETVAL));

        semctl(flag, 0, SETVAL, 0); // 设置flag为0，允许semaphore2写
        printf("flag = %d\n", semctl(flag, 0, GETVAL));
    }
    return 0;
}
```

3. semaphore2.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>

int main(){
    int flag = semget(0x123, 1, 0);

    while(1) {
        while(semctl(flag, 0, GETVAL) != 0); // 等待flag为0

        FILE *fp = fopen("out.txt", "a");
        fprintf(fp, "894 PROC2 MYFILE2\n");
        fclose(fp);
        printf("flag = %d\n", semctl(flag, 0, GETVAL));

        semctl(flag, 0, SETVAL, 1); // 设置flag为1，允许semaphore1写
        printf("flag = %d\n", semctl(flag, 0, GETVAL));
```

```
    }
    return 0;
}
```

semaphore1.c 在信号量值为 1 时进行操作，并将信号量值设置为 0。
semaphore2.c 在信号量值为 0 时进行操作，并将信号量值设置为 1。
运行`vi sem1.c`创建文件
运行`./sem1`创建信号量
运行`gcc -g -o semaphore1 semaphore1.c`，`gcc -g -o semaphore2 semaphore2.c`编译文件
当 flag 未设置成 1 时 semaphore2.c 被阻塞，程序 semaphore1 中 flag 设置成 1 时 semaphore2 执行，而后 semaphore1 被阻塞，如此交替循环。out.txt 被交替写入，通过`cat out.txt`查看写入内容。

三、关键句的含义

1. while(semctl(flag, 0, GETVAL) != 1)

进程等待信号量 `flag` 的值变为 1，然后才能继续执行。这里使用了 `semctl` 函数来获取信号量的值。

2. semctl(flag, 0, SETVAL, 0)

设置信号量 `flag` 的值为 0，表示让另一个进程可以开始执行。这里使用了 `semctl` 函数来设置信号量的值。

四、证明当进程 A 占用 semaphore 后，进程 B 想要占用 semaphore 时进入睡眠

当 flag 未设置成 1 时 semaphore2.c 被阻塞，程序 semaphore1 中 flag 设置成 1 时 semaphore2 执行，而后 semaphore1 被阻塞，如此交替循环。out.txt 被交替写入。

Semaphore2 执行到' while(semctl(flag, 0, GETVAL) != 0); // 等待 flag 为 0'时因为此时 flag 为 1，被阻塞，执行 Semaphore1，当 flag 置 1 时 Semaphore2 可以继续执行。out.txt 被交替写入。

五、strict alternation 算法中的`turn`变量是否需要加锁

严格轮换算法中的 `turn` 变量是一个标志，用来指示当前轮到哪个进程执行。在严格

轮换算法中，`turn` 变量是交替被读取和写入的，但是在单核系统中，读取和写入一个整数是原子操作，因此不需要额外的锁。

在多核系统中，如果多个核心同时访问 `turn` 变量，可能会出现竞态条件。因此，在多核系统中，为了保证严格轮换算法的正确性，需要使用同步原语（如互斥锁、原子操作等）来保护 `turn` 变量的读写操作。