实验六 信号量实现进程同步

实验目的

进程同步是操作系统多进程/多线程并发执行的关键之一，进程同步是并发进程为了完成共同任务采用某个条件来协调他们的活动，这是进程之间发生的一种直接制约关系。本次试验是利用信号量进行进程同步。

实验软件环境

Ubuntu 21.04

实验内容

用信号量解决生产者-消费者问题. 本例中**生产者和消费者数量可能大于一，需要解决竞争冒险问题**。

• 生产者进程生产产品，消费者进程消费产品。
• 当生产者进程生产产品时，如果没有空缓冲区可用，那么生产者进程必须等待消费者进程释放出一个缓冲区。
• 当消费者进程消费产品时，如果缓冲区中没有产品，那么消费者进程将被阻塞，直到新的产品被生产出来。

不是很看得懂助教发的ppt，通过man semctl查阅手册，发现实验ppt里提到的那些函数是System-V信号量的函数，具体内容可以Google一下。而<sys/mman.h>这个头文件是用于使用共享主存的，而共享主存的实验还没做

值得注意的是，System-V信号量 semget与semctl是分开的，因此创建信号量并初始化不是一个原子操作，可能出现信号量未初始化就被使用引起的错误。本例中可以先创建信号量再fork来避免这种情况的发生。

对两个信号量进行PV操作的顺序很关键，对于消费者来说，若先对mutexid P再对fullid P，可能会出现共享缓冲区全为空，消费者拿到了互斥锁后一直等待fullid却等不到的情况。因此需要将fullid放在mutexid之前，确保能对缓冲区进行操作后再修改互斥信号量。

但这样写经过几次测试后也陷入了死锁，部分输出如下，在这段输出之前消费者A进程已经结束：

```
A waiting for fullid
the producer set number 99
Prod waiting for empty
B waiting for mutexid
Prod waiting for mutex
the consumer B get number 99
B waiting for fullid
the producer set number 100
Produce finished
A waiting for mutexid
the consumer A get number 100
the sum is 5050
consumer A is over
B waiting for mutexid
the consumer B get number 96
B waiting for fullid

```

B拿到99后生产者生产了100,随后mutexid和fullid给A拿到了，A拿到100。随后B不知为何拿到了fullid（本该为0的），

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/sem.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAXSEM 5

typedef struct Shared_Data {
    int array[5], sum, get, set;
}Data;

int fullid, emptyid, mutexid, fid;  //信号量
struct sembuf P, V;
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
               (Linux-specific) */
};
int new_sem(int val) {  //创建一个初值为val的信号量
    union semun arg;
    arg.val = val;
    int semid = semget(IPC_PRIVATE, 1, IPC_CREAT | 0666);
    if (semctl(semid, 0, SETVAL, arg) == -1) {
        puts("error semctl new_sem");
        return -1;
    }
    return semid;
}
int del_sem(int semid) {
    union semun arg;
    if (semctl(semid, 0, IPC_RMID) == -1) {
        puts("error del_sem");
        return -1;
    }
    return 0;
}

int main() {
    struct Shared_Data *addr;
    P.sem_flg = SEM_UNDO;
    P.sem_num = 0;
    P.sem_op = -1;
    V.sem_flg = SEM_UNDO;
    V.sem_num = 0;
```

```
    V.sem_op = 1;
    //由于进行的是具有亲缘关系的进程间通信，因此可以将fd指定为-1,进行匿名映射，
    //此时须指定flags参数中的MAP_ANON
    addr = mmap(NULL, sizeof(Data), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANON, -1, 0);
    fullid = new_sem(0);
    emptyid = new_sem(MAXSEM);
    mutexid = new_sem(1);
    //fid = new_sem(1);
    int pid = fork();
    if (pid < 0) {
        puts("fork err");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {  //子进程
        puts("Current: Parent");
        int i = 0;
        while (i < 100) {
            puts("Prod waiting for empty");
            semop (emptyid, &P, 1);
            puts("Prod waiting for mutex");
            semop (mutexid, &P, 1);
            addr->array[addr->set%MAXSEM] = i+1;
            printf("the producer set number %d\n", addr->array[addr-
>set%MAXSEM]);
            addr->set++;
            i++;    //下发的ppt有误，i++操作应当放在互斥区内，随后应该对fullid进行V
操作
            semop (fullid, &V, 1);
            semop (mutexid, &V, 1);
        }
        //sleep(3);
        puts("Produce finished");
        exit(EXIT_SUCCESS);
    } else {
        int pida = fork();
        if (pida < 0) {
            puts("fork err");
            exit(EXIT_FAILURE);
        } else if (pida == 0) {
            puts("Current: CA");
            while (1) {
                printf("A %d\n", addr->get);
                if (addr->get >= 100) {
                    break;
                }
                //semop (fid, &P, 1);
                puts ("A waiting for fullid");
                semop (fullid, &P, 1);
                puts ("A waiting for mutexid");
                semop (mutexid, &P, 1);
                //semop (fid, &V, 1);
                if (addr->get == 100) break;
                addr->sum += addr->array[addr->get % MAXSEM];
                printf("the consumer A get number %d\n", addr->array[addr-
>get % MAXSEM]);
                addr->get++;
                if (addr->get == 100) {
                    printf("the sum is %d\n", addr->sum);
```

```
                    semop (emptyid, &V, 1);
                    //semop (mutexid, &V, 1);
                    break;
                }
                semop (emptyid, &V, 1);
                semop (mutexid, &V, 1);
            }
            //sleep(2);
            puts("consumer A is over");
            exit(EXIT_SUCCESS);
        } else {
            int pidb = fork();
            if (pidb < 0) {
                puts("fork err");
                exit(EXIT_FAILURE);
            } else if (pidb == 0) {
                puts("Current: CB");
                while (1) {
                    printf("B %d\n", addr->get);
                    if (addr->get >= 100) {
                        break;
                    }
                    puts("B waiting for fullid");
                    semop (fullid, &P, 1);
                    puts("B waiting for mutexid");
                    semop (mutexid, &P, 1);
                    addr->sum += addr->array[addr->get % MAXSEM];
                    printf("the consumer B get number %d\n", addr-
>array[addr->get % MAXSEM]);
                    addr->get++;
                    if (addr->get == 100) {
                        printf("the sum is %d\n", addr->sum);
                        semop (emptyid, &V, 1);
                        //semop (mutexid, &V, 1);
                        break;
                    }
                    semop (emptyid, &V, 1);
                    semop (mutexid, &V, 1);
                }
                puts("consumer B is over");
                //sleep(2);
                exit(EXIT_SUCCESS);
            } else {
                while ((waitpid(-1, NULL, 0)) > 0);
                assert(del_sem(fullid) == 0);
                assert(del_sem(emptyid) == 0);
                assert(del_sem(mutexid) == 0);
                //assert(del_sem(fid) != 0);
                munmap(addr, sizeof(Data));
                exit(EXIT_SUCCESS);
            }
        }
    }
    return 0;
}

```