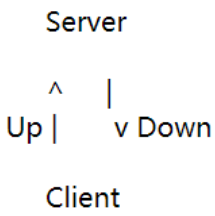
Lab07 Assignment

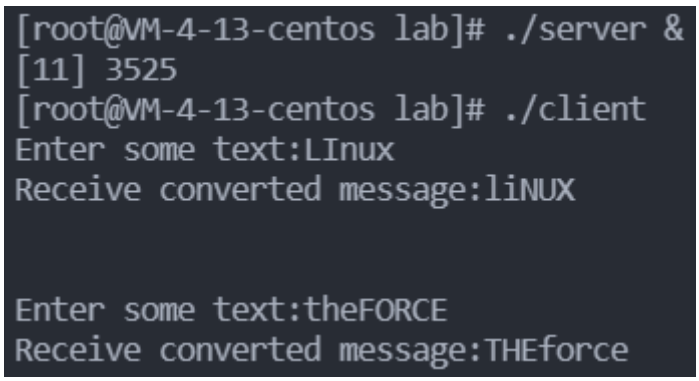
> 班级：
> 学号：
> 姓名：

1. 请实现这样一个程序

请实现这样一个程序：客户端进程（Client）和服务器进程（Server）通过**消息队列**进行通信，消息队列共有两个，Up 和 Down，如下图所示：



客户端进程接受用户从终端的输入，并通过 Up 消息队列将消息传递给服务器进程，然后等待服务器进程从 Down 消息队列传回消息。服务器进程从 Up 接收到消息后**将大小写字母转换**，并通过 Down 传回给客户端进程，客户端随后输出转换后的消息。（例如：客户端通过 Up 发送'linuX', 将从 Down 接收到'LINUx'）。多个客户端同时使用 Up 和 Down 消息队列时也应该能够正常工作，因此需要使用消息类型 mtype 区分来自不同客户端的消息。要求程序输出如下的效果：




```c
//client code
#ifndef MSGQUE_EXAMP
#define MSGQUE_EXAMP
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/stat.h>
#define MAX_TEXT 512
#define MSG_KEY_UP 335
#define MSG_KEY_DOWN 336

struct my_msg_st
{
    long my_msg_type;
```

```
    char text[MAX_TEXT];
};
#endif

int main(){
    int msgid_up,msgid_down;
    msgid_up = msgget((key_t)MSG_KEY_UP, IPC_CREAT|0660);
    msgid_down = msgget((key_t)MSG_KEY_DOWN, IPC_CREAT|0660);
    if(msgid_down == -1 || msgid_up == -1){
        perror("get message queue failed ");
        return -1;
    }

    while(1) {
        struct my_msg_st snd_data,rcv_data;
        snd_data.my_msg_type=getpid();
        printf("Enter some text:");
        fgets(snd_data.text,MAX_TEXT,stdin);
        if (msgsnd(msgid_up, (void *)&snd_data, MAX_TEXT, 0) == -1){
            perror("msgsnd failed ");
            return -1;
        }
        if (msgrcv(msgid_down,(void *)&rcv_data, MAX_TEXT,getpid(),0) == -1){
            perror("msgrcv failed ");
            return -1;
        }
        printf("Receive converted message:%s\n\n",rcv_data.text);
    }
    exit(0);
}
```

```
//server code
#ifndef MSGQUE_EXAMP
#define MSGQUE_EXAMP
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/stat.h>
#define MAX_TEXT 512
#define MSG_KEY_UP 335
#define MSG_KEY_DOWN 336

struct my_msg_st
{
    long my_msg_type;
    char text[MAX_TEXT];
};
#endif

int main(int argc,char **argv) {
    int msgid_up,msgid_down;
    msgid_up = msgget((key_t)MSG_KEY_UP, IPC_CREAT|0660);
```

```
    msgid_down = msgget((key_t)MSG_KEY_DOWN, IPC_CREAT|0660);
    if(msgid_down == -1 || msgid_up == -1){
        perror("get message queue failed ");
        return -1;
    }

    while (1) {
        struct my_msg_st snd_data,rcv_data;
        if (msgrcv(msgid_up,(void *)&rcv_data, MAX_TEXT,0,0) == -1){
            perror("msgrcv failed ");
            return -1;
        }
        char tmp[MAX_TEXT];
        strcpy(tmp,rcv_data.text);

        for(int i=0;i<strlen(tmp);i++){
            if(tmp[i]>='A' && tmp[i]<='Z')
                tmp[i]+=32;
            else if(tmp[i]>='a' && tmp[i]<='z')
                tmp[i]-=32;
        }

        strcpy(snd_data.text,tmp);
        snd_data.my_msg_type=rcv_data.my_msg_type;
        if (msgsnd(msgid_down, (void *)&snd_data, MAX_TEXT, 0) == -1){
            perror("msgsnd failed ");
            return -1;
        }
    }
}
```

2. 请实现这样一个程序

请实现这样一个程序：一个进程创建 3 个子进程A、B、C，每个子进程都打印你的学号，但要求每个进程都打印完这一位数字后，才能有进程开始下一位数字的打印，并且进程打印顺序按照进程A、B、C依次打印，**在打印的数字前加上A、B、C**。例如，我的学号是`19373075`，那么输出结果应该是`A1B1C1A9B9C9A3B3C3A7B7C7A3B3C3A0B0C0A7B7C7A5B5C5`。仅允许使用**信号量**作为同步工具。

```
//code
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>
#include <unistd.h>
#include <semaphore.h>
#include <fcntl.h>


char SEM_NAME1[]= "process1";
char SEM_NAME2[]= "process2";
char SEM_NAME3[]= "process3";
char id[]="19373075";

int main(int argc,char **argv) {
    pid_t pid;
```

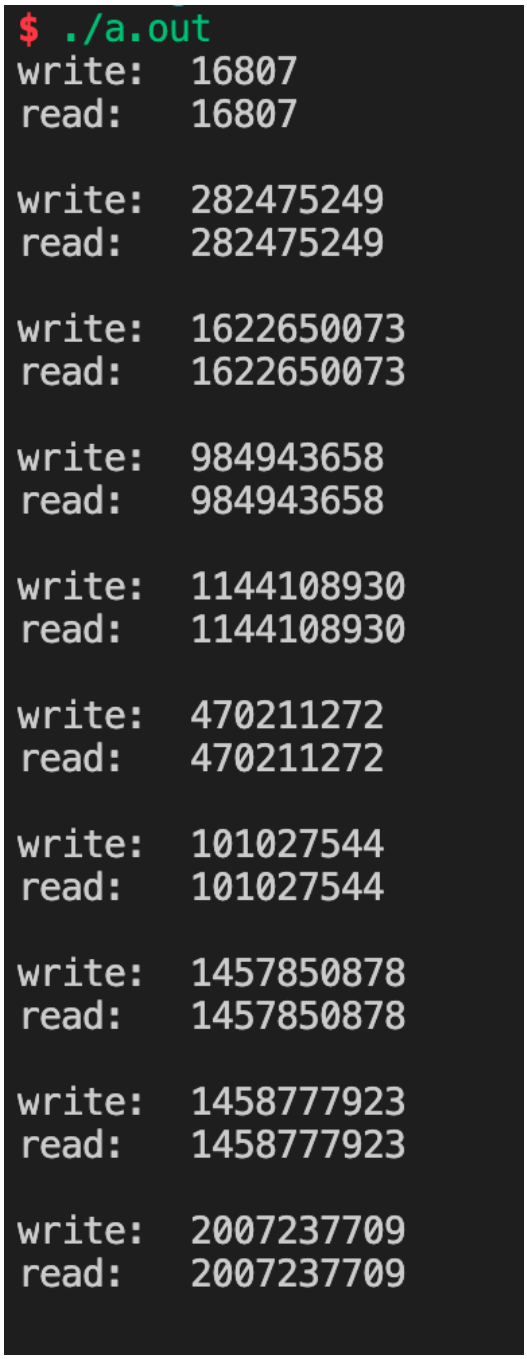
```
    sem_t *sem1,*sem2,*sem3;
    int i,j;
    sem1 = sem_open(SEM_NAME1,O_CREAT,0777,1);
    sem2 = sem_open(SEM_NAME2,O_CREAT,0777,0);
    sem3 = sem_open(SEM_NAME3,O_CREAT,0777,0);

    if(sem1 == SEM_FAILED||sem2==SEM_FAILED||sem3==SEM_FAILED) {
        perror("unable to execute semaphore");
        sem_close(sem1);
        sem_close(sem2);
        sem_close(sem3);
        exit(-1);
    }

    for(i=0;i<3;i++){
        pid=fork();
        if(pid==0)break;
    }

    if(i==0){
        for(j=0;j<8;j++){
            sem_wait(sem1);
            printf("A%c",id[j]);fflush(stdout);
            sem_post(sem2);
        }
        exit(0);
    }
    else if(i==1){
        for(j=0;j<8;j++){
            sem_wait(sem2);
            printf("B%c",id[j]);fflush(stdout);
            sem_post(sem3);
        }
        exit(0);
    }
    else if(i==2){
        for(j=0;j<8;j++){
            sem_wait(sem3);
            printf("C%c",id[j]);fflush(stdout);
            sem_post(sem1);
        }
        exit(0);
    }
    else{
        for(j=0;j<3;j++){
            wait(0);
        }

    }
    sem_close(sem1);
    sem_close(sem2);
    sem_close(sem3);
    sem_unlink(SEM_NAME1);
    sem_unlink(SEM_NAME2);
    sem_unlink(SEM_NAME3);
```

```
    exit(0);
}
```

注意：编译时加上 -pthread

3. 请实现这样一个程序

在《Linux 编程基础》一书对共享内存的讲解中，其给出的例子是一个进程向共享内存写，然后终止，然后再启动一个进程从共享内存中读。请实现这样一个程序：同时使用**信号量**和**共享内存**实现一个这样的功能，同时运行两个进程A和B，A进程向共享内存中写入数据后阻塞，等待B进程读，读完之后A再写，然后B再读……。要求程序输出如下的效果：



一共要求输出 10 组，30 行，`read` 行之后有一空行，以便于明显区分组别；`write` 和 `read` 后面的数字请不要显示明显的规律性，请使用 `rand()` 函数获取，并一定在调用 `rand()` 函数之前，使用 `srand(unsigned int seed)` 重置随机种子，其中，`seed` 为你的学号。

```
//code
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/sem.h>
#include<sys/types.h>
#include<unistd.h>
#include<string.h>
#include<errno.h>

union semu{
    int val;
    struct semid_ds* buf;
    unsigned short* array;
    struct seminfo* _buf;
};

int set_semvalue(int s_id,int index,int value){
    union semu su;
    su.val=value;
    if(semctl(s_id,index,SETVAL,su)==-1)
        return 0;
    return 1;
}
int P(int s_id,int index){
    struct sembuf ss;
    ss.sem_num=index;
    ss.sem_op=-1;
    ss.sem_flg=SEM_UNDO;
    if(semop(s_id,&ss,1)==-1){
        perror("P error!");
        return 0;
    }
    return 1;
}
int V(int s_id,int index){
    struct sembuf ss;
    ss.sem_num=index;
    ss.sem_op=1;
    ss.sem_flg=SEM_UNDO;
    if(semop(s_id,&ss,1)==-1){
        perror("V error!");
        return 0;
    }
    return 1;
}
int delete_sem(int s_id){
    union semu su;
    if(semctl(s_id,0,IPC_RMID,su)==-1||semctl(s_id,1,IPC_RMID,su)==-1)
        return 0;
    return 1;
}

int main(int argc, char const *argv[])
{
    int shm_id;
```

```
    struct shmid_ds buf;
    key_t key=ftok("./",0);
    int* smap;
    if(key==-1){
        perror("ftok error!");
        return -1;
    }
    shm_id=shmget(key,sizeof(int),0664|IPC_CREAT);
    if(shm_id==-1){
        perror("shmget error!");
        return -1;
    }
    smap=(int*)shmat(shm_id,NULL,0);

    int sem=semget(key,2,0664|IPC_CREAT);
    if(sem==-1){
        perror("semget error!");
        return -1;
    }
    if(!(set_semvalue(sem,0,1)&&set_semvalue(sem,1,0))){
        perror("sem init error!");
        return -1;
    }

    if(fork()==0){
        srand(19373075);
        int i,tmp;
        for(i=0;i<10;i++){
            P(sem,0);
            tmp=rand();
            *smap=tmp;
            printf("Write:%d\n",tmp);
            V(sem,1);
        }
    }
    else{
        int i,tmp;
        for(i=0;i<10;i++){
            P(sem,1);
            tmp=*smap;
            printf("Read :%d\n\n",tmp);
            V(sem,0);
        }
        if(shmdt(smap)==-1){
            perror("detach share memory error!");
            return -1;
        }
        shmctl(shm_id,IPC_RMID,&buf);
        delete_sem(sem);
    }

    return 0;
}
```