进程间通信

1. 共享内存

1.1 特点

1）共享内存是一种最为高效的进程间通信方式，进程可以直接读写内存，而不需要任何数据的拷贝

2）为了在多个进程间交换信息，内核专门留出了一块内存区，可以由需要访问的进程将其映射到自己的私有地址空间

3）进程就可以直接读写这一内存区而不需要进行数据的拷贝，从而大大提高的效率。

4）由于多个进程共享一段内存，因此也需要依靠某种同步机制，如互斥锁和信号量等

1.2 步骤

0）创建 key 值

1）创建或打开共享内存

2）映射

3）取消映射

4）删除共享内存

1.3 函数接口

key_t ftok(const char *pathname, int proj_id);

功能：创建 key 值

参数：pathname：文件名

proj_id：取整型数的低 8 位数值

返回值：成功：key 值

失败：-1

int shmget(key_t key, size_t size, int shmflg);

功能：创建或打开共享内存

参数：

key 键值

size 共享内存的大小

shmflg IPC_CREAT|IPC_EXCL|0777

返回值：成功 shmid

出错 -1

void *shmat(int shmid,const void *shmaddr,int shmflg);

功能：映射共享内存，即把指定的共享内存映射到进程的地址空间用于访问

参数：

shmid 共享内存的 id 号

shmaddr 一般为 NULL，表示由系统自动完成映射

如果不为 NULL，那么有用户指定

shmflg：SHM_RDONLY 就是对该共享内存只进行读操作

0 可读可写

返回值：成功：完成映射后的地址，

出错：-1 的地址

用法：if((p = (char *)shmat(shmid，NULL,0)) == (char *)-1)

int shmdt(const void *shmaddr);

功能：取消映射

参数：要取消的地址

返回值：成功 0

失败的-1

int shmctl(int shmid,int cmd,struct shmid_ds *buf);

功能：(删除共享内存)，对共享内存进行各种操作

参数：

shmid 共享内存的 id 号

cmd IPC_STAT 获得 shmid 属性信息，存放在第三参数

IPC_SET 设置 shmid 属性信息，要设置的属性放在第三参数

IPC_RMID:删除共享内存，此时第三个参数为 NULL 即可

返回：成功 0

查看共享内存的命令：

```
ipcs  -m
```

删除共享内存的命令：

```
ipcrm -m  shmid
```

代码案例：

```
int main(int argc, char const *argv[])
{
    key_t key;
    int shmid;
    //创建 key 值
    key = ftok("./app", 'b');
    if (key < 0)
    {
        perror("ftok err");
        return -1;
    }
    printf("%#x\n", key);
    //创建或打印共享内存
    shmid = shmget(key, 128, IPC_CREAT | IPC_EXCL | 0666);
    if (shmid < 0)
    {
        if (errno == EEXIST)
            shmid = shmget(key, 128, 0666);
        else
        {
            perror("shmget err");
            return -1;
        }
    }
    printf("shmid:%d\n", shmid);
    //映射
    char *p = NULL;
    p = shmat(shmid, NULL, 0); //NULL:系统自动进行映射 0:可读可写
    if(p == (char *)-1)
    {
        perror("shmat err");
        return -1;
    }
    strcpy(p, "hello");
    printf("%s\n", p);
    //取消映射
    shmdt(p);
    //删除共享内存
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

练习：一个进程从终端输入，另一个进程将数据输出，借助共享内存通信。

要求：当输入 quit 时程序退出

同步：标志位

2. 信号灯集

2.1. 特点：

信号灯(semaphore)，也叫信号量。它是不同进程间或一个给定进程内部不同线程间同步的机制；System V 的信号灯是一个或者多个信号灯的一个集合。其中的每一个都是单独的计数信号灯。而 Posix 信号灯指的是单个计数信号灯。

通过信号灯集实现共享内存的同步操作

2.2 步骤：

0）创建 key 值

1）创建或打开信号灯集 semget

2）初始化信号灯集 semctl

3）pv 操作 semop

4）删除信号灯集 semctl

2.3 函数接口

```
int semget(key_t key, int nsems, int semflg);
```

功能：创建/打开信号灯

参数：key：ftok 产生的 key 值

nsems：信号灯集中包含的信号灯数目

semflg：信号灯集的访问权限，通常为 IPC_CREAT |0666

返回值：成功：信号灯集 ID

失败：-1

```
int semop ( int semid, struct sembuf  *opsptr,  size_t  nops);
```

功能：对信号灯集合中的信号量进行 PV 操作

参数：semid：信号灯集 ID

opsptr:操作方式

nops: 要操作的信号灯的个数 1 个

返回值：成功 ：0

失败：-1

```
struct sembuf {
   short  sem_num; // 要操作的信号灯的编号
   short  sem_op;  //    0 :  等待，直到信号灯的值变成 0
                   //   1  :  释放资源，V 操作
                   //   -1 :  分配资源，P 操作
    short  sem_flg; // 0（阻塞）,IPC_NOWAIT, SEM_UNDO
};
```

用法：

申请资源 P 操作：

```
    mysembuf.sem_num = 0;
    mysembuf.sem_op = -1;
    mysembuf.sem_flg = 0;
    semop(semid, &mysembuf, 1);
```

释放资源 V 操作：

```
    mysembuf.sem_num = 0;
    mysembuf.sem_op = 1;
    mysembuf.sem_flg = 0;
    semop(semid, &mysembuf, 1);
```

```
int semctl ( int semid, int semnum,  int cmd…/*union semun
arg*/);
```

功能：信号灯集合的控制（初始化/删除）

参数：semid：信号灯集 ID

semnum：要操作的集合中的信号灯编号

cmd：

GETVAL：获取信号灯的值，返回值是获得值

2.4 命令

ipcs -s:查看信号灯集

ipcrm -s semid：删除信号灯集

例子：

```
union semun {
    int val; //信号灯的初值
};
int main(int argc, char const *argv[])
{
    key_t key;
    int semid;
    key = ftok("./app", 'b');
    if (key < 0)
    {
        perror("ftok err");
        return -1;
    }
    printf("%#x\n", key);
    //创建或打开信号灯集
    semid = semget(key, 2, IPC_CREAT | IPC_EXCL | 0666);
    if (semid < 0)
    {
        if (errno == EEXIST)
            semid = semget(key, 2, 0666);
        else
        {
            perror("semget err");
            return -1;
        }
    }
    else
    {
        //初始化
        union semun sem;
        sem.val = 10;
        semctl(semid, 0, SETVAL, sem); //对编号为0的信号灯初值设置为
10
        sem.val = 0;
        semctl(semid, 1, SETVAL, sem); //对编号为1的信号灯初值设置为
0
    }
    printf("semid:%d\n", semid);
    printf("%d\n",semctl(semid, 0, GETVAL));//获取编号为0的信号灯的
值
    printf("%d\n",semctl(semid, 1, GETVAL));//获取编号为1的信号灯的
值
```