

# **Система межпроцессного взаимодействия ІРС**

# Общие концепции

Необходимые заголовочные файлы и прототип

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

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

```
key_t ftok ( char * filename, char proj )
```

## Параметры

**filename** — строка, содержащая имя файла

**proj** — добавочный символ

# Общие концепции

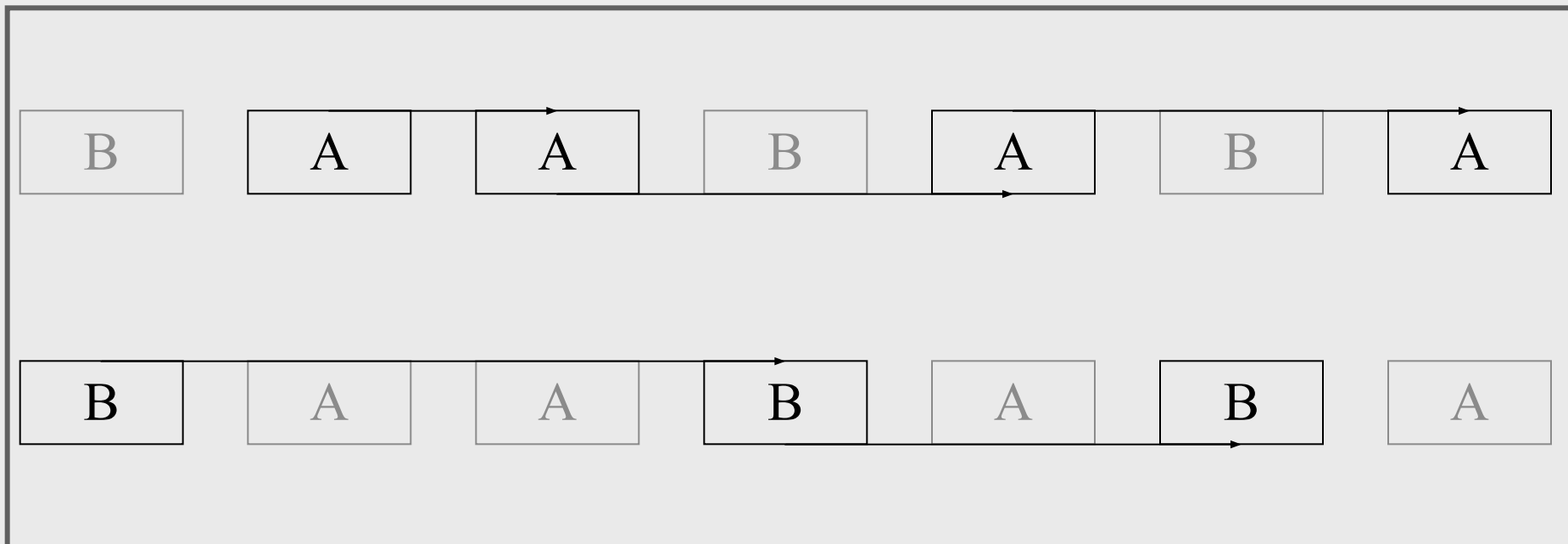
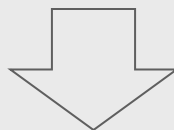
- **<ResName>get ( key, ..., flags )** —  
создание/подключение
- Флаги создания/подключения:
  - **IPC\_PRIVATE** (доступность только породившему процессу)
  - **IPC\_CREAT** (создать новый или подключиться к существующему)
  - **IPC\_EXCL** ( + IPC\_CREAT создание только нового)
  - ...

# IPС: очередь сообщений

1. Общие концепции
2. Создание/доступ к очереди сообщений
3. Отправка сообщений
4. Получение сообщений
5. Управление очередью сообщений
6. Пример. Использование очереди сообщений
7. Пример. Очередь сообщений .Модель «клиент-сервер»

# Очередь сообщений

- Организация очереди сообщений по принципу **FIFO**
- Использование типов сообщений



# Создание/доступ к очереди сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/message.h>
```

```
int msgget ( key_t key, int msgflag )
```

## Параметры

**key** — ключ

**msgflag** — флаги, управляющие поведением вызова

## Возвращаемое значение

В случае успеха возвращается положительный дескриптор очереди, в случае неудачи возвращается  $-1$ .

# Отправка сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgsnd ( int msqid, const void * msgp, size_t msgsz,  
int msgflg )
```

## Параметры

**msqid** — дескриптор очереди, полученный в результате вызова msgget()

**msgp** — указатель на буфер:

- **long msgtype** — тип сообщения
- **char msgtext[]** — данные (тело сообщения)

# Отправка сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgsnd ( int msqid, const void * msgp, size_t msgsz,  
int msgflg )
```

## Параметры

**msgsz** — размер тела сообщения

**msgflg**

- 0 процесс блокируется, если для отправки сообщения недостаточно системных ресурсов
- **IPC\_NOWAIT** — работа без блокировки (возврат -1)



# Получение сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgrecv ( int msqid, void * msgp, size_t msgsz, long  
msgtyp, int msgflg )
```

## Параметры

**msqid** — дескриптор очереди

**msgp** — указатель на буфер

**msgsz** — размер тела сообщения

**msgtyp** — тип сообщения, которое процесс желает  
получить

# Получение сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgrecv ( int msqid, void * msgp, size_t msgsz, long  
msgtyp, int msgflg )
```

## Параметры

**msgflg** — побитовое сложение флагов

- **IPC\_NOWAIT** — если сообщения в очереди нет, то возврат **-1**
- **MSG\_NOERROR** — разрешение получать сообщение, даже если его длина превышает емкость буфера

## Возвращаемое значение

В случае успеха возвращает количество прочитанных байтов в теле сообщения.

# Управление очередью сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgctl ( int msqid, int cmd, struct msgid_ds * buf )
```

## Параметры

**msqid** — дескриптор очереди

**cmd** — команда

- **IPC\_STAT** — скопировать структуру, описывающую управляющие параметры очереди по адресу, указанному в параметре **buf**
- **IPC\_SET** — заменить структуру, описывающую управляющие параметры очереди, на структуру, находящуюся по адресу, указанному в параметре **buf**
- **IPC\_RMID** — удалить очередь

# Управление очередью сообщений

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/msg.h>
```

```
int msgctl (int msqid, int cmd, struct msgid_ds *buf)
```

## Параметры

**buf** — структура, описывающая параметры очереди.

Тип **msgid\_ds** описан в заголовочном файле `<sys/message.h>`, и представляет собой структуру, в полях которой хранятся права доступа к очереди, статистика обращений к очереди, ее размер и т.п.

## Возвращаемое значение

В случае успеха возвращается 0.

# Использование очереди сообщений

## Основной процесс

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/message.h>
#include <stdio.h>
struct
{
    long mtype ;
    char Data [ 256 ] ;
} Message ;
```

```
int main ( int argc, char ** argv )
{
    key_t key;
    int msgid;
    char str [ 256 ] ;

    key = ftok ( "/usr/mash", 's' ) ;
    msgid = msgget ( key, 0666 |
                    IPC_CREAT | IPC_EXCL ) ;
    for ( ; ; )
    {
        gets ( str ) ;
        strcpy ( Message . Data, str ) ;
        ...
    }
}
```

# Использование очереди сообщений

## Основной процесс

```
...
switch ( str [ 0 ] ) {
    case 'a' : case 'A' :
        Message . mtype = 1 ;
        msgsnd ( msgid, ( struct
                    msgbuf * ) ( &
Message ),          strlen (
str ) + 1, 0 ) ;
        break ;
    case 'b' : case 'B' :
        Message . mtype = 2 ;
        msgsnd(msgid, ( struct
                    msgbuf * ) ( &
Message ),          strlen (
str ) + 1, 0 ) ;
        break ;
```

```
case 'q' : case 'Q' :
    Message . mtype = 1 ;
    msgsnd ( msgid, ( struct msgbuf * )
        ( & Message ), strlen ( str ) + 1,
0 ) ;

    Message . mtype = 2 ;
    msgsnd ( msgid, ( struct msgbuf * )
        ( & Message ), strlen ( str ) + 1,
0 ) ;

    sleep ( 10 ) ;
    msgctl ( msgid, IPC_RMID, NULL
    ) ;

    exit ( 0 ) ;

default :
    break ;
}
```

# Использование очереди сообщений

## Процесс-приемник А

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

struct {
    long mtype ;
    char Data [ 256 ] ;
} Message ;

int main ( int argc, char ** argv )
{
    key_t key ;
    int msgid ;

    key = ftok ( "/usr/mash", 's' ) ;
    msgid = msgget ( key, 0666 ) ;
    for ( ; ; ) {
        msgrcv ( msgid, ( struct msgbuf * )
            ( & Message ), 256, 1, 0 ) ;
        printf ( "%s", Message . Data ) ;
        if ( Message . Data [ 0 ] == 'q' || Message .
            Data [ 0 ] == 'Q' )
            break ;
    }
    exit () ; }
```

# Пример: «Клиент-сервер»

## Сервер

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <string.h>

int main ( int argc, char ** argv )
{
    struct {
        long mstype ;
        char mes [ 100 ] ;
    } message ;

    struct {
        long mstype ;
        long mes ;
    } messagefrom ;

    key_t key ;
    int mesid ;

    key = ftok ( "example", 'r' ) ;
    mesid = msgget ( key, 0666 | IPC_CREAT |
                    IPC_EXCL ) ;

    while ( 1 )
    {
        msgrcv ( mesid, & messagefrom, sizeof (
            messagefrom ) – sizeof ( long ), 1, 0 ) ;
        message . mstype = messagefrom . mes ;
        strcpy ( message . mes, “Message for client”
        ) ;
        msgsnd ( mesid, & message, sizeof
            ( message ) – sizeof ( long ), 0 ) ;
    }

    return 0 ;
}
```



# Пример: «Клиент-сервер»

## Клиент

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int main ( int argc, char ** argv )
{
    struct {
        long mstype;
        long mes;
    } message;
    struct {
        long mstype;
        char mes[100];
    } messagefrom;

    key_t key;
    int mesid;

    long pid = getpid () ;
    key = ftok ( "example", 'r' ) ;
    mesid = msgget ( key, 0666 ) ;
    message.mstype = 1 ;
    message.mes = pid ;
    msgsnd ( mesid, & message, sizeof
(messaget) – sizeof ( long ), 0 ) ;
    msgrcv ( mesid, & messagefrom, sizeof (
messagefrom ) – sizeof ( long ), pid, 0 ) ;
    printf ( "%s", messagefrom.mes ) ;
    return 0 ;
}
```

# IPC: разделяемая память

## Создание общей памяти

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

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

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

```
int shmget ( key_t key, int size, int shmflg )
```

### Параметры

**key** — ключ для доступа к разделяемой памяти

**size** — размер области памяти

**shmflg** — флаги управляющие поведением вызова

### Возвращаемое значение

дескриптор области памяти, в случае неудачи — -1.

# Доступ к разделяемой памяти

Необходимые заголовочные файлы и прототип

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

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

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

```
char * shmat ( int shmid, char * shmaddr, int shmflg )
```

## Параметры

**shmid** — дескриптор области памяти

**shmaddr** — «адрес подключения» - адрес, начиная с которого необходимо подсоединить разделяемую память (>0 или =0)

**shmflg** — флаги, например, **SHM\_RDONLY** подсоединяемая область будет использоваться только для чтения

## Возвращаемое значение

Возвращает адрес, начиная с которого будет отображаться присоединяемая разделяемая память. При неудаче возвращается -1.

# Отключение от разделяемой памяти

Необходимые заголовочные файлы и прототип

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

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

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

```
int shmdt ( char * shmaddr )
```

## Параметры

**shmaddr** — адрес прикрепленной к процессу памяти, который был получен при вызове `shmat()`

## Возвращаемое значение

В случае успешного выполнения функция возвращает 0, в случае неудачи — -1.

# Управление разделяемой памятью

Необходимые заголовочные файлы и прототип

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

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

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

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

## Параметры

**shmid** — дескриптор области памяти

**cmd:**

- **IPC\_STAT** — скопировать структуру, описывающую управляющие параметры области памяти
- **IPC\_SET** — заменить структуру, описывающую управляющие параметры области памяти, на структуру, находящуюся по адресу, указанному в параметре **buf**.
- **IPC\_RMID** удалить
- **SHM\_LOCK, SHM\_UNLOCK** — заблокировать или разблокировать область памяти.

**buf** — структура, описывающая управляющие параметры области памяти.

# Пример. Работа с общей памятью в рамках одного процесса

```
int main ( int argc, char ** argv )
{
    key_t key;
    char * shmaddr ;

    key = ftok ( “/tmp/ter”, 'S' ) ;
    shmid = shmget ( key, 100, 0666 | IPC_CREAT | IPC_EXCL ) ;
    shmaddr = shmat ( shmid, NULL, 0 ) ;
    putm ( shmaddr ) ;
    .....
    shmctl ( shmid, IPC_RMID, NULL ) ;
    exit () ;
}
```

**IPC: массив семафоров**

# Создание/доступ к семафору

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/sem.h>
```

```
int semget ( key_t key, int nsems, int semflag )
```

## Параметры

**key** — уникальный идентификатор ресурса

**nsems** — количество семафоров

**semflag** — флаги

## Возвращаемое значение

Возвращает целочисленный дескриптор созданного разделяемого ресурса, либо -1, если ресурс не удалось создать.



# Операции над семафором

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/sem.h>
```

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

## Параметры

**semid** — дескриптор ресурса

**cmd\_buf** — массив из элементов типа **sembuf**

**nops** — количество элементов в массиве **cmd\_buf**

# Операции над семафором

Значение семафора с номером **num** равно **val**.

```
struct sembuf
{
    short sem_num ; /* номер семафора    */
    short sem_op ;   /* операция        */
    short sem_flg ;  /* флаги операции   */
}
```

**Если**  $\text{sem\_op} \neq 0$  **то**

**пока**  $(\text{val} + \text{sem\_op} < 0)$  [процесс блокирован]

$\text{val} = \text{val} + \text{sem\_op}$

**Если**  $\text{sem\_op} = 0$  **то**

**пока**  $(\text{val} \neq 0)$  [процесс блокирован]

    [возврат из вызова]

# Управление массивом семафоров

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/sem.h>
```

```
int semctl (int semid, int num, int cmd, union semun arg)
```

## Параметры

**semid** — дескриптор массива семафоров

**num** — номер семафора в массиве

**cmd** — операция

- **IPC\_SET** изменить значение, параметры семафора
- **IPC\_RMID** удалить массив семафоров
- и др.

**arg** — управляющие параметры

## Возвращаемое значение

Возвращает значение, соответствующее выполнявшейся операции (по умолчанию 0), в случае неудачи — -1

# Управление массивом семафоров

Необходимые заголовочные файлы и прототип

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

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

```
#include <sys/sem.h>
```

```
int semctl (int semid, int num, int cmd, union semun arg)
```

```
<sys/sem.h>
```

```
union semun
```

```
{
```

```
    int val ; /* значение одного семафора */
```

```
    struct semid_ds * buf ; /* параметры массива семафоров в  
        целом (количество, права доступа, статистика доступа)*/
```

```
    unsigned short * array ; /* массив значений семафоров */
```

```
}
```

# **Пример. Использование разделяемой памяти и семафоров**

# Использование разделяемой памяти и семафоров

## 1-ый процесс

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#define NMAX 256
```

```
int main ( int argc, char ** argv )
{
    key_t key;
    int semid, shmid ;
    struct sembuf sops ;
    char * shmaddr ;
    char str [ NMAX ] ;

    key = ftok ( “/usr/ter/exmpl”, 'S' ) ;
    semid = semget ( key, 1, 0666 | IPC_CREAT
| IPC_EXCL ) ;
    shmid = shmget ( key, NMAX, 0666 |
IPC_CREAT | IPC_EXCL ) ;
    shmaddr = shmat(shmid, NULL, 0 ) ;
```

# Использование разделяемой памяти и семафоров

## 1-ый процесс

```
...
semctl ( semid, 0, SETVAL, (int) 0 ) ;
sops . sem_num = 0 ;
sops . sem_flg = 0 ;
do
{
    printf( “Введите строку:” ) ;
    if ( fgets ( str, NMAX, stdin ) ==
        NULL )    strcpy ( str, “Q” ) ;
    strcpy ( shmaddr, str ) ;

sops . sem_op = 3 ;
semop ( semid, & sops, 1 ) ;
sops . sem_op = 0 ;
semop ( semid, & sops, 1 ) ;
}
while ( str [ 0 ] != ‘Q’ ) ;
shmdt ( shmaddr ) ;
shmctl ( shmid, IPC_RMID,  NULL);
semctl ( semid, 0, IPC_RMID, (int) 0 ) ;
return 0 ;
}
```

# Использование разделяемой памяти и семафоров

## 2-ой процесс

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#define NMAX 256

int main ( int argc, char ** argv )
{
    key_t key ;
    int semid, shmid ;
    struct sembuf sops ;
    char * shmaddr ;
    char str [ NMAX ] ;

    key = ftok ( “/usr/ter/exmpl”, 'S' ) ;
    semid = semget ( key, 1, 0666 ) ;
    shmid = shmget ( key, NMAX, 0666 ) ;
    shmaddr = shmat ( shmid, NULL, 0 ) ;
    sops.sem_num = 0 ;
```



# Использование разделяемой памяти и семафоров 2-ой процесс

```
sops.sem_flg = 0;
do {
    printf ( “Waiting... \n” );
    sops . sem_op = -2 ;
    semop ( semid, & sops, 1 ) ;
    strcpy ( str, shmaddr ) ;
    if ( str [ 0 ] == ‘Q’ )
        shmdt ( shmaddr ) ;
    sops . sem_op = -1 ;
    semop ( semid, & sops, 1 ) ;
    printf ( “Read from shared memory: %s\n”, str ) ;
} while ( str [ 0 ] != ‘Q’ ) ;
return 0 ;
}
```