Система межпроцессного взаимодействия IPC.

Система межпроцессного взаимодействия IPC. Состав.

- Очереди сообщений
- Семафоры
- Разделяемая память

Общие концепции

- Для именования объекта IPC используется уникальный ключ, по которому процессу возвращается дескриптор объекта
- Для каждого IPC-ресурса поддерживается идентификатор его владельца и структура, описывающая
  - права доступа к нему (только две категории прав доступа- по чтению и по записи).
  - информацию о создателе и владельце ресурса, их группе
  - его ключ.

<sys/ipc.h>

struct ipc_perm

Общие концепции

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

key_t ftok (char *filename, char proj)

filename – строка, содержащая имя файла

proj – добавочный символ (может использоваться, например, для поддержания разных версий программы)

Общие концепции

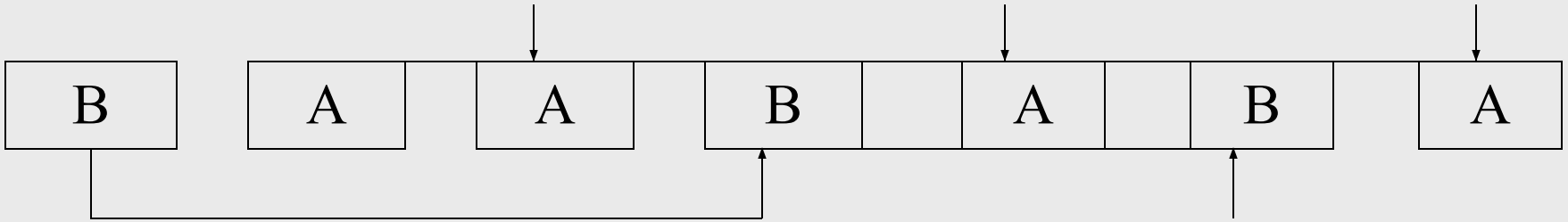
- Создание/подключение разделяемого ресурса по заданному ключу. **IPC_PRIVATE**
- Флаги создания/подключения:
  - **IPC_CREAT**
  - **IPC_EXCL**
- Значения переменной **errno** при ошибках:
  - **ENOENT**
  - **EEXIST**
  - **EACCESS**
  - **ENOSPC**

IPC: очередь сообщений.

Очередь сообщений

- Организация очереди сообщений по принципу **FIFO**

- Использование типов сообщений



Создание/доступ к очереди сообщений

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

#include <sys/message.h>

int msgget (key_t key, int msgflag)

key – ключ

msgflag – флаги, управляющие поведением вызова

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

Отправка сообщений

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

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

msqid – идентификатор очереди, полученный в результате вызова msgget()

msgp – указатель на буфер следующей структуры: **long msgtype** **-**тип сообщения

char msgtext[ ] -данные (тело сообщения)

Отправка сообщений

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

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

msgsz –размер буфера (не более определенной в заголовочном файле <sys/msg.h> константы MSGMAX)

msgflg = 0 вызывающий процесс блокируется, если для посылки сообщения недостаточно системных ресурсов =
IPC_NOWAIT выход из вызова немедленно, возврат -1

В случае успеха возвращается 0

Получение сообщений

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

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

msqid – идентификатор очереди

msgp – указатель на буфер

msgsz – размер буфера

msgtyp тип сообщения, которое процесс желает получить

= 0 любого типа

> 0 типа msgtyp

< 0 наименьшее значение среди типов, которые меньше модуля msgtyp

Получение сообщений

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

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

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

В случае успеха возвращается 0

Управление очередью сообщений

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

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

msgid – идентификатор ресурса

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

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

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

buf– структура, описывающая параметры очереди.

Тип msgid_ds описан в заголовочном файл <sys/message.h>, и представляет собой структуру, в полях которой хранятся права доступа к очереди, статистика обращений к очереди, ее размер и т.п.

В случае успеха возвращается 0

Пример. Использование очереди сообщений.

Программа, в которой основной процесс читает некоторую текстовую строку из стандартного ввода и в случае, если строка начинается с буквы 'a', то эта строка в качестве сообщения будет передана процессу A, если 'b' - процессу B, если 'q' - то процессам A и B, затем будет осуществлен выход. Процессы A и B распечатывают полученные строки на стандартный вывод.

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, chr **argv)
{
    key_t key;
    int msgid;
    char str[256];

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

```
    ...
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;
}   }   }
```

Процесс-приемник A

```
#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, chr **argv)
{
    key_t key;
    int msgid;

    key = ftok("/usr/mash", 's');
    msgid = msgget(key, 0666 | IPC_CREAT)
    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. Очередь сообщений. Модель «клиент-сервер».

Server

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

```
int main(int argc, chr **argv)
{
struct {
    long mestype;
    char mes [100];
} messageto;
struct {
    long mestype;
    long mes;
 } messagefrom;
key_t key;
int mesid;
...
```

Server

```
…
key = ftok("example", 'r');
mesid = msgget (key, 0666 | IPC_CREAT);
while(1) {
    if (msgrcv(mesid, &messagefrom,
    sizeof(messagefrom), 1, 0) < 0)
     break;
    messageto.mestype = messagefrom.mes;
    strcpy( messageto.mes, "Message for client");
    msgsnd (mesid, &messageto, sizeof(messageto),  0);
}
msgctl (mesid, IPC_RMID, 0);
return 0;          }
```

Client

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

```
int main(int argc, chr **argv)
{
struct {
    long mestype;
    long mes;
} messageto;
struct {
    long mestype;
    char mes[100];
} messagefrom;
key_t key;
int mesid;
...
```

Client

```
...
long pid = getpid();
key = ftok("example", 'r');
mesid = msgget (key, 0);
messageto.mestype = 1;
messageto.mes = pid;
msgsnd (mesid, &messageto,
    sizeof(messageto), 0);
while ( msgrcv (mesid, &messagefrom,
    sizeof(messagefrom), pid, IPC_NOWAIT) <
    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 shmemflg)

key – ключ для доступа к разделяемой памяти

size – размер области памяти

shmeflg – флаги управляющие поведением вызова

В случае успешного завершения вызов возвращает положительное число – дескриптор области памяти, в случае неудачи - -1.

Доступ к разделяемой памяти

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

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

shmid – дескриптор области памяти

shmaddr – виртуальный адрес в адресном пространстве, начиная с которого необходимо подсоединить разделяемую память (чаще всего =0, то есть выбор редоставляется системе)

shmflg – флаги, например, **SHM_RDONLY** подсоединяемая область будет использоваться только для чтения.

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

Открепление разделяемой памяти

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt(char *shmaddr)
```

shmaddr - адрес прикрепленной к процессу памяти, который был получен при вызове shmat()

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

Управление разделяемой памятью

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

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

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 – блокировать или разблокировать область памяти.

buf – структура, описывающая управляющие параметры области памяти.

Пример. Работа с общей памятью в рамках одного процесса.

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

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

IPC: массив семафоров.

Схема использования семафоров

- С каждым разделяемым ресурсом связывается один семафор из набора
- Значение >0 – ресурс свободен, <0 – ресурс занят
- Перед обращением к ресурсу процесс уменьшает значение соответствующего семафора
- Закончив работу с ресурсом, процесс увеличивает значение семафора
- В случае реализации взаимного исключения используется двоичный семафор.

Создание/доступ к семафору

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

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

key – ключ

sems – количество семафоров

semflag – флаги, определяющие права доступа и те операции, которые должны выполняться (открытие семафора, проверка, и т. д.).

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

Операции над семафором

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

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

semid – идентификатор ресурса

semop – указатель на структуру, определяющую операции, которы
нужно призвести над семафором

nops– количество указателей на эту структуру, которые передаютс
функцией semop() (операций может быть несколько и
операционная система гарантирует их атомарное выполнение).

Операции над семафором

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

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

Если semop ≠ 0 **то** **если** val+semop < 0 **то** **пока** !(val+semop≥0) [процесс стоит] val=val+semop

Если semop = 0 **то** **если val** ≠ 0 **то** **пока** (val ≠ 0) [процесс стоит]
[возврат из вызова]

Управление массивом семафоров

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

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

semid – дескриптор массива семафоров

num – индекс семафора в массиве

cmd – операция

IPC_SET заменить управляющие наборы семафоров на те, которые указаны в **arg.buf**

IPC_RMID удалить массив семафоров

и др.

arg – управляющие параметры

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

Управление массивом семафоров

```
#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)*/
    ushort *array;  /* массив значений семафоров */
}
```

Пример. 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);
    shmid = shmget(key, NMAX, 0666 | IPC_CREAT);
    shmaddr = shmat(shmid, NULL, 0);
    …
```

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;}
```

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 | IPC_CREAT);
    shmid = shmget(key, NMAX, 0666 | IPC_CREAT);
    shmaddr = shmat(shmid, NULL, 0);
    sops.sem_num = 0;
    …
```

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;
}
```