Оглавление

ЛАБОРАТОРНАЯ РАБОТА ОЧЕРЕДИ СООБЩЕНИЙ (IPC)

Теория

Ключи key_t и функция ftok

В таблице (см. **Ошибка! Источник ссылки не найден.**) было отмечено, что в именах трех типов System V IPC использовались значения **key_t**. Заголовочный файл <sys/types.h> определяет тип key_t как целое (32-разрядное). Значения переменным этого типа обычно присваиваются функцией ***ftok***.

Функция ***ftok*** преобразовывает существующее полное имя и целочисленный идентификатор в значение типа **key_t**, называемое ключом IPC (IPC key):

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

key_t ftok(const char *pathname, int id);
//Возвращает ключ IPC либо -1 при возникновении ошибки
```

На самом деле функция использует полное имя файла, и младшие 8 бит идентификатора для формирования целочисленного ключа IPC.

Эта функция действует в предположении, что для конкретного приложения, использующего IPC, клиент и сервер используют одно и то же полное имя объекта IPC, имеющее какое-то значение в контексте приложения. Это может быть:

- имя файла данных, используемого сервером;
- или имя еще какого-нибудь объекта файловой системы.

Если клиенту или серверу для связи требуется только один канал IPC, идентификатору можно присвоить, например, значение 1. Если требуется

несколько каналов IPC (например, один от сервера к клиенту и один в обратную сторону), идентификаторы должны иметь разные значения: например, 1 и 2. После того как клиент и сервер договорятся о полном имени и идентификаторе, они оба вызывают функцию ***ftok*** для получения одинакового ключа IPC.

Если указанное полное имя не существует или недоступно вызывающему процессу, ***ftok*** возвращает значение -1.

Помните, что файл, имя которого используется для вычисления ключа, не должен создаваться и удаляться сервером в процессе работы, поскольку каждый раз при создании заново файл получает другой номер индексного дескриптора. И мы можем получить другой ключ, возвращаемый функцией ***ftok*** при очередном вызове.

Структура ipc_perm

Для каждого объекта IPC, как для обычного файла, в ядре хранится структура:

```
struct ipc_perm {
        uid_t uid; // идентификатор пользователя владельца
        gid_t gid; // идентификатор группы владельца
        uid_t cuid; // идентификатор пользователя создателя
        gid_t cgid; // идентификатор группы создателя
        mode_t mode; // разрешения чтения/записи
        ulong_t seq; // последовательный номер канала
        key_t key; // ключ IPC
                };
```

Эта структура вместе с другими переименованными константами для функций System V IPC определена в файле <sys/ipc.h>. Далее поля этой структуры будут рассмотрены более подробно.

Создание и открытие System V IPC

Три функции ***get***<>, используемые для создания или открытия объектов IPC (см. **Ошибка! Источник ссылки не найден.**), принимают ключ **key_**t в качестве одного из аргументов и возвращает целочисленный идентификатор.

У приложения есть две возможности задания ключа первого аргумента функций ***get***<>:

- Вызвать ***ftok*** и передать ей полное имя и идентификатор;
- Указать в качестве ключа константу IPC_PRIVATE, гарантирующую создание нового уникального объекта IPC.

Последовательность действий приведена на рисунке Рис. 1 Определение идентификаторов IPC по ключам.

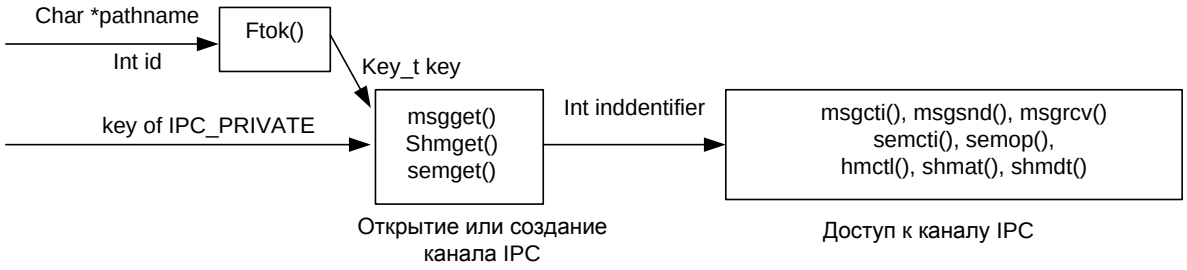

Рис. 1 Определение идентификаторов IPC по ключам

Все три функции ***get***<> принимают в качестве второго аргумента набор флагов oflag, задающий биты разрешений чтения/записи (поле mode структуры ipc_perm) для объекта IPC и определяющий, создается ли новый объект IPC или производится обращение к уже существующему объекту. Используются следующие правила:

- Ключ IPC_PRIVATE гарантирует создание уникального объекта IPC. Никакие возможные комбинации полного имени и идентификатора не могут привести к тому, что функция ***ftok*** вернет в качестве ключа значение IPC_PRIVATE.

- Установка бита IPC_CREAT аргумента oflag приводит к созданию новой записи для указанного ключа, если она еще не существует. Если же обнаруживается существующая запись, возвращается ее идентификатор.

Одновременная установка битов IPC_CREAT и IPC_EXCL и аргумента oflag приводит к созданию новой записи для указанного ключа только в том случае, если такая запись еще не существует. Если же обнаруживается существующая запись, функция возвращает ошибку EEXIST (объект IPC уже существует).

Логическая диаграмма последовательности действий при открытии объекта IPC изображена рисунке (Рис. 2 Открытие объекта IPC).

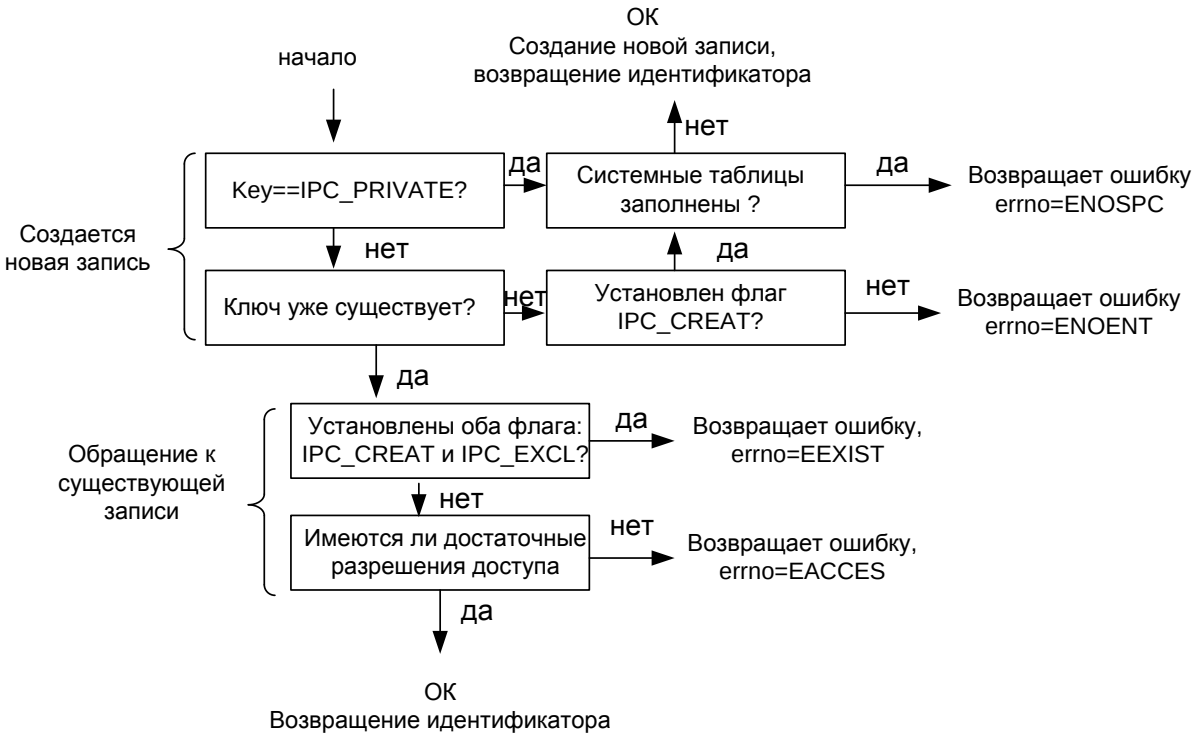

Рис. 2 Открытие объекта IPC

При создании нового объекта IPC с помощью одной из функций ***get***<>, вызванной с флагом IPC_CREAT, в структуру ipc_perm заносится следующая информация:

- часть битов аргумента oflag задают значение поля mode структуры ipc_perm. В таблице (Таблица 1 Значения mode для разрешений чтения/запись) приведены биты разрешений для трех типов IPC (запись >> 3 означает сдвиг вправо на три бита);
- поля cuid и cgid получают значения, равные действующим идентификаторам пользователя и группы вызывающего процесса. Эти два поля называются идентификаторами создателя;
- поля uid и gid структуры ipc_perm также устанавливаются равными действующим идентификаторам вызывающего процесса. Эти два поля называются идентификаторами владельца.

Число (восьмеричное)	Очередь сообщений	Семафор	Разделяемая память	Описание
0400	MSG_R	SEM_R	SHM_R	Пользователь-чтение
0200	MSG_W	SEM_A	SHM_W	Пользователь-запись
0040	MSG_R>>3	SEM_R>>3	SHM_R>>3	Группа-чтение
0020	MSG_W>>3	SEM_A>>3	SHM_W>>3	Группа-запись
0004	MSG_R>>6	SEM_R>>6	SHM_R>>6	Прочие-чтение
0002	MSG_W>>6	SEM_A>>6	SHM_W>>6	Прочие-запись

Таблица 1 Значения mode для разрешений чтения/запись

Очереди сообщений

Очередь сообщений – это заголовок, указывающий на связанный список сообщений (см. Рис. 3 Структура очереди сообщений System V).

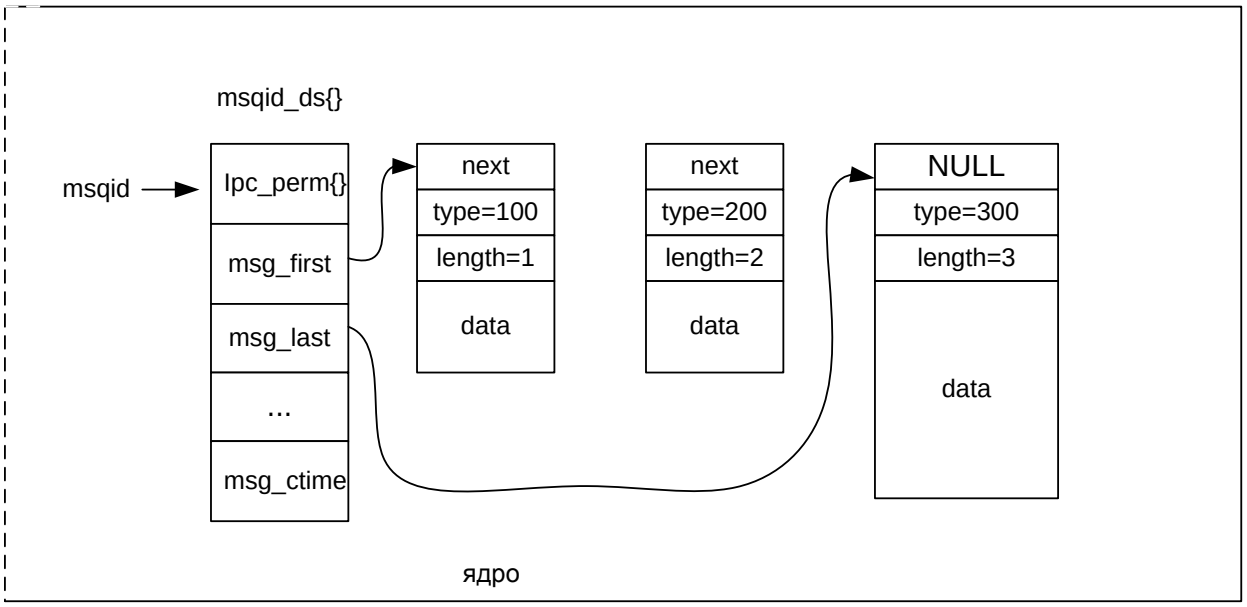

Рис. 3 Структура очереди сообщений System V

Каждое сообщение содержит 32–разрядную переменную типа, следующую за областью данных. Процесс создает или получает очередь сообщений при помощи системного вызова ***msgget***:

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

msgpid = ***msgget***(key, flag);

Здесь key – это целое число, задаваемое пользователем. Для создания новой очереди сообщений необходимо указать флаг IPC_CREAT. Задание флага IPC_EXCL ведет к ошибочному завершению работы вызова в том случае, если очередь с указываемым ключом уже существует. Переменная msgpid используется в дальнейших вызовах для доступа к очереди.

Для того чтобы поместить сообщение в очередь, необходимо использовать:

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

msgsnd(msgqid, msgp, count, flag);

Здесь msqid является дескриптором объекта, полученного в результате вызова ***msgget.*** Параметр msgp указывает на буфер, содержащий тип сообщения и его данные, размер которого равен msgsz байт. Буфер имеет следующие поля:

- long msgtype – тип сообщения
- char msgtext[] - данные сообщения

Если msgtyp равен 0, функция ***msgrcv*** получит первое сообщение из очереди. Если величина msgtype больше 0, будет получено первое сообщение указанного типа. Если меньше 0, функция ***msgrcv*** получит сообщение с минимальным значением типа, меньше или равного абсолютному значению msgtype.

Флаг IPC_NOWAIT используется для возврата ошибки, если сообщение невозможно отправить без блокировки(например, когда очередь переполнена, так очередь обычно обладает настраиваемым ограничением на количество хранящихся в ней данных).

На рисунке (**Ошибка! Источник ссылки не найден.**) показаны операции над очередями сообщений. Каждая очередь описывается в виде строки в таблице ресурсов очередей сообщений.

struct msqid_ds {
struct ipc_perm msg_perm; /* разрешения */
struct msg* msg_first; /*указатель на первое сообщение в очереди*/
struct msg*msg_last; /*указатель на последнее сообщение в очереди*/

```
ushort msg_cbytes; /*размер очереди в байтах*/
ushort msg_qnum; /*количество сообщений в очереди*/
ushort msg_qbytes; /*максимально допустимый размер очереди в байтах*/
…
};
```

Сообщения располагаются внутри очереди в порядке их поступления. Они удаляются из очереди по принципу “первым вошел, первым вышел” при чтении их процессом при помощи вызова:

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

count=msgrcv(msgqid, msgp, maxcnt, msgtype, flag);
```

msgp указывает на буфер, в который помещается входящее сообщение, maxcnt ограничивает максимально прочитываемое количество байтов. Если входящее сообщение длиннее, чем maxcnt, то оно будет обрезано. Пользователь должен быть уверен в том, что буфер, указанный при помощи msgp, имеет достаточный объем для хранения maxcnt байтов данных. Возвращаемая функцией величина указывает на успешно прочитанное количество байтов.

Чему нужно научиться

Изучить использование очередей сообщений System V.

Задание А

Уровень 1 (А)

Посмотрите на Скелет кода для задания 1 (А) **и разберитесь, что там происходит.**

Уровень 2 (А)

Напишите программы для сервера и клиента. Одновременно могут работать несколько клиентов.

Сервер должен включать обработчик сигнала SIGINT (с восстановлением диспозиции и удалением очереди сообщений системным вызовом ***msgctl***() для корректного завершения сервера при получении сигнала SIGINT). Создайте очередь сообщений, используя системный вызов ***msgget***(key, PERM | IPC_CREAT).

Сервер в цикле ожидает сообщение, а затем читает его из очереди (тип 1) и посылает на каждое сообщение ответ клиенту (тип 2). Выводите сообщения на экран для проверки. Если происходит ошибка, используете функцию kill() (отправьте сигнал SIGINT).

Client.c должен получить доступ к очереди сообщений, а затем отправить сообщение серверу (тип 1). Клиент ожидает сообщение, а затем читает его (тип 2). Выведите сообщение на экран для проверки.

Для чтения и посылки сообщения используйте системные вызовы ***msgrcv***() и ***msgsnd***().

Откомпилируйте программы (gcc –o server server.c; gcc –o client client.c), запустите на выполнение исполняемые модули ./server и ./client, отправьте процессу server сигнал SIGINT.

Создайте несколько клиентов, запустите их одновременно.

Уровень 3 (А)

Посмотрите на Скелет кода для задания 3 (А) **и разберитесь, что там происходит и допишите недописанные функции.**

Скелет кода для задания 1 (А)

mesg.h

```
#define MAXBUFF 80
#define PERM 0666
```

```
/*Определим структуру сообщения*/
typedef struct our_msgbuf {
        long mtype;
        char buff[MAXBUFF];
        } Message;
```

Server.c

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

main()
{
        Message message;
        key_t key;
        int msgid, length, n;

        /*Получим ключ*/
        if((key = ftok(“server”, ‘A’))<0)
        { printf(“Невозможно получить ключ\n”); exit(1); }

        /*Тип принимаемых сообщений*/
        message.mtype = 1L;

        /*Создадим очередь сообщений*/
        if((msgid = msgget(key,PERM|IPC_CREAT))<0)
        { printf(“Невозможно создать очередь\n”); exit(1); }

        /*Примем сообщение*/
```

```
        n = msgrcv(msgid, &message, sizeof(message), message.mtype, 0);

        /*Если сообщение поступило, выведем его содержимое на терминал*/
        if(n>0){
                if(write(1, message.buff,n) != n){ printf("Ошибка вывода\n");
exit(1);}
                }
        else { printf("Ошибка чтения сообщения\n"); exit(1); }
        /*Удалить очередь должен клиент*/
        exit(0);
}
```

Client.c

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

main()
{

        Message message;
        key_t key;
        int msgid, length;

        /*Тип посылаемого сообщения*/
        message.mtype = 1L;

        /*Получим ключ*/
```

```
        if((key = ftok(“server”, ‘A’))<0)
        { printf(“Невозможно получить ключ\n”); exit(1); }

        /*Получим доступ к очереди сообщений*/
        /*очередь уже должна быть создана сервером*/
        if((msgid = msgget(key, 0))<0)
        { printf(“Невозможно получить доступ кочереди\n”); exit(1); }

        /*Подготоим сообщение*/
        if ((length = sprintf(message.buff,“IPC Messages!\n”))<0)
        { printf(“Ошибка копирования в буфер\n”); exit(1); }

        /*Передадим сообщение*/
        if(msgsnd(msgid, (void*) &message, length, 0) != 0)
        { printf(“Ошибка записи сообщения в очередь\n); exit(1); }

        /*Удалим очередь сообщений*/
        if(msgctl(msgid, IPC_RMID, 0)<0)
        { printf(“Ошибка удаления очереди\n”); exit(1);}
        exit(0);
}
```

Скелет кода для задания 3 (А)

client_msg.c

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h>
```

```
#define MQ_KEY1 1234L
#define MQ_KEY2 2345L
#define MAXMESGDATA 4096

struct mymesg {
        long mesg_len;
        long mesg_type;
        char mesg_data[MAXMESGDATA];
        };

void client(int readid, int writeid);
ssize_t mesg_send(int id, struct mymesg *mptr);
ssize_t mesg_recv(int id, struct mymesg *mptr);

int main(int argc,char **argv)
{
        int readid, writeid;

        if( (writeid=msgget(MQ_KEY1,0666)) <0)
        {
                printf("Client: can not get writeid!\n"); exit(1);
        }
        printf("Client:writeid=%d\n",writeid);

        if((readid=msgget(MQ_KEY2,0666)) <0)
        {
                printf("Client: can not get readid!\n"); exit(1);
        }
        printf("Client: readid=%d\n",readid);
```

```
        client(readid,writeid);

        if((msgctl(readid,IPC_RMID, NULL)) < 0)
        {
                printf("Client: can not delete massage queue2!\n"); exit(1);
        }

        if((msgctl(writeid,IPC_RMID, NULL)) <0)
        { printf("Client: can not delete massage queue1!\n"); exit(1); }

        exit(0);
}

void client(int readid, int writeid)
{
        size_t len;
        ssize_t n;
        struct mymesg ourmesg;

        printf("Client:readid=%d writeid=%d\n",readid,writeid);

        fgets(ourmesg.mesg_data,MAXMESGDATA, stdin);
        len=strlen(ourmesg.mesg_data);

        if(ourmesg.mesg_data[len-1]=='\n') len--;
        ourmesg.mesg_len=len;

        ourmesg.mesg_type=1;

        printf("Client: %s\n",ourmesg.mesg_data);
```

```
        mesg_send(writeid,&ourmesg);

        printf("Client: before recv!\n");

        while((n= mesg_recv(readid, &ourmesg))>0)
                write(1,ourmesg.mesg_data, n);
}

ssize_t mesg_send(int id, struct mymesg *mptr)
{
        …
}

ssize_t mesg_recv(int id, struct mymesg *mptr)
{
        …
}
```

Server_msg.c

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h>

#define MQ_KEY1 1234L
#define MQ_KEY2 2345L
#define MAXMESGDATA 4096
```

```
struct mymesg {
        long mesg_len;
        long mesg_type;
        char mesg_data[MAXMESGDATA];
        };

void server(int readid, int writeid);
ssize_t mesg_send(int id, struct mymesg *mptr);
ssize_t mesg_recv(int id, struct mymesg *mptr);

int main(int argc,char **argv)
{
        int readid, writeid;
        key_t key1, key2;

        printf("Server: HELLO!\n");

/*
        if((key1=ftok("/home/tvk/IPC/input.txt",'A'))<0)
        {
                printf("Server: can not get key!\n"); exit(1);
        }
        printf("key1=%x\n",key1);
*/
        if((readid=msgget(MQ_KEY1, 0666|IPC_CREAT))<0)
        {
                printf("Server: can not get readid!\n"); exit(1);
        }
        printf("Server: readid=%d\n",readid);
```

```
/*
        if((key2=ftok("/home/tvk/IPC/server_msg.c",'B'))<0)
        {
                printf("Server: can not get key!\n"); exit(1);
        }
        printf("key2=%x\n",key2);
*/
        if((writeid=msgget(MQ_KEY2, 0666|IPC_CREAT))<0)
        {
                printf("Server: can not get readid!\n"); exit(1);
        }
        printf("Server: writeid=%d\n",writeid);

        server(readid,writeid);

        exit(0);
}

void server(int readid, int writeid)
{
        FILE *fp;
        ssize_t n;
        struct mymesg ourmesg;

        printf("Server:readid=%d writeid=%d\n",readid,writeid);

        ourmesg.mesg_type=1;

        if( (n=mesg_recv(readid, &ourmesg)) == 0)
        { printf("Server: can not read file name\n"); exit(1); }
```

```
        ourmesg.mesg_data[n]='\0';

        printf("Server: file name %s\n",ourmesg.mesg_data);

        if( (fp=fopen(ourmesg.mesg_data,"r"))==NULL)
        { printf("Server: can not open file name\n"); }
        else
        {
                printf("Server: %s is opened\n",ourmesg.mesg_data);

        while(fgets(ourmesg.mesg_data, MAXMESGDATA,fp) != NULL)
                {
                        ourmesg.mesg_len=strlen(ourmesg.mesg_data);
                        printf("Server: %s\n",ourmesg.mesg_data);
                        mesg_send(writeid,&ourmesg);
                }
        }
        fclose(fp);
        ourmesg.mesg_len=0;
        mesg_send(writeid,&ourmesg);
}

ssize_t mesg_send(int id, struct mymesg *mptr)
{
        …
}

ssize_t mesg_recv(int id, struct mymesg *mptr)
{
        …
```

```
}
```