Сергей Юрьевич Шилов

# Системное программирование для современных платформ

# Системное программирование для современных платформ

## 7. System V IPC

# Системное программирование для современных платформ

System V IPC:

- SysV R1: 1983
- очереди сообщений, семафоры, разделяемая память
- замена сигналов, пайпов, работы с файлами через отображение в память
- polling (message-base) programming

# Системное программирование для современных платформ

Пространство имён объектов IPC представляет собой множество хэшей (ключей, токенов), генерируемых функцией ftok. Хэш генерируется на основе двух параметров: inod'a реально существующего файла и числа, называемого идентификатором проекта. В каждом семействе IPC объектов (очередей сообщений, семафоров, областей разделяемой памяти) собственное пространство имён. Функции, работающие непосредственно с IPC объектами принимают в качестве аргументов дескриптор соответствующего объекта, который предварительно должен быть получен из ключа, аналогично файловому дескриптору.

# Системное программирование для современных платформ

Схема работы с IPC-объектами

1)Определите подходящий ключ IPC для использования с ftok(3).
2)Поставьте IPC-специфичный идентификатор, связанный с ключом IPC, используя для этого msgget(2), semget(2) или shmget(2) для очередей сообщений, семафоров или разделяемой памяти соответственно.
3)Измените свойства экземпляра IPC при помощи msgctl(2), semctl(2) или shmctl(2).
4)Используйте конкретный экземпляр IPC.
5)В конце уничтожьте IPC-экземпляр при помощи msgctl(2), semctl(2) или shmctl(2) и флажка IPC_RMID

# Системное программирование для современных платформ

```
# include <sys/types.h>
# include <sys/ipc.h>
key_t ftok(const char *pathname, int proj_id);
```

**ОПИСАНИЕ**

Функция **ftok** использует файл с именем **pathname** (которое должно указывать на существующий файл к которому есть доступ) и младшие 8 бит **proj_id** (который должен быть отличен от нуля) для создания ключа с типом **key_t**, используемого в **System V IPC** для работы с **msgget(2)**, **semget(2)**, **и shmget(2)**.
Возвращаемое значение одинаково для всех имен, указывающих на один и тот же файл при одинаковом значении **proj_id**. Возвращаемое значение должно отличаться, когда (одновременно существующие) файлы или идентификаторы проекта различаются.

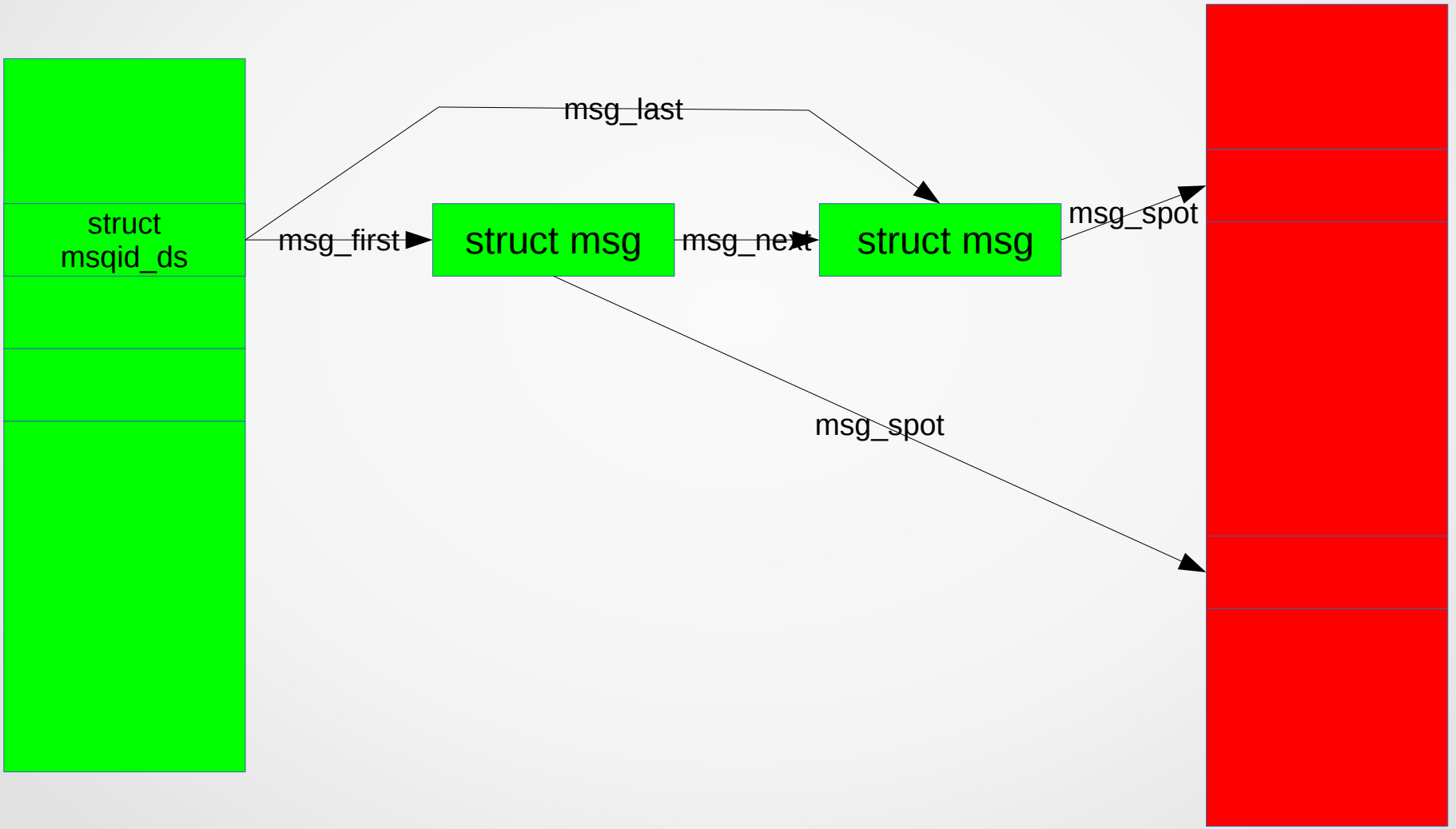
**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

В случае удачного завершения вызова возвращается значение созданного ключа **key_t**. При ошибке возвращается **-1**, а в переменную **errno** записывается код ошибки согласно системному вызову **stat(2)**.

# Системное программирование для современных платформ

## Очереди сообщений SysV IPC

# Системное программирование для современных платформ



# Системное программирование для современных платформ

## Системные переменные

| | | |
|---|---|---|
| MSGMNI | Максимальное число очередей в системе в любой момент времени | ошибка |
| MSGMAX | Максимальная длина сообщения (в байтах) | ошибка |
| MSGMNB | Максимальный объём всех сообщений (в байтах) | блокировка |
| MSGTQL | Максимальное число сообщений во всех очередях | блокировка |

## API

| | | |
|---|---|---|
| msgget | Открытие или создание очереди и получение её дескриптора | open |
| msgsnd | Передача сообщения в очередь | write |
| msgrcv | Приём сообщения из очереди | read |
| msgctl | Управление параметрами и закрытие очереди | stat, unlink, chmod, chown |

# Системное программирование для современных платформ

Структуры данных

```
struct msgbuf {    /* буфер сообщения для вызовов msgsnd и msgrcv*/
    long mtype;                                    /* тип сообщения */
    char *mtext;};                                /* текст сообщения */

struct msg {
   struct msg *msg_next;          /* следующее сообщение в очереди */
   long   msg_type;
   char  *msg_spot;                        /* адрес текста сообщения */
   short  msg_ts;};                                /* размер текста */

struct ipc_perm {  /*структура описания владельца и прав доступа*/
   key_t  key;
   ushort uid;                              /* euid и egid владельца */
   ushort gid;
   ushort cuid;                             /* euid и egid создателя */
   ushort cgid;
   ushort mode;          /* режим доступа, см. режимные флаги ниже */
   ushort seq;};          /* порядковый номер использования гнезда */
```

# Системное программирование для современных платформ

```
struct msqid_ds {                /*структура msqid для каждой очереди*/
   struct ipc_perm msg_perm;
   struct msg *msg_first;              /* первое сообщение в очереди */
   struct msg *msg_last;            /* последнее сообщение в очереди */
   time_t msg_stime;               /* время последнего вызова msgsnd */
   time_t msg_rtime;               /* время последнего вызова msgrcv */
   time_t msg_ctime;                   /* время последнего изменения */
   struct wait_queue *wwait;
   struct wait_queue *rwait;
   ushort msg_cbytes;
   ushort msg_qnum;
   ushort msg_qbytes;    /* максимальное число байтов на очередь */
   ushort msg_lspid;       /* pid последнего испустившего msgsnd */
   ushort msg_lrpid;};                    /* последний полученный pid */
```

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

**ОПИСАНИЕ**

Эта функция возвращает идентификатор очереди сообщений, связанный со значением параметра **key** . Она также создает новую очередь сообщений, если **key** равен **IPC_PRIVATE**; в случае если **key** не равен **IPC_PRIVATE**, то с параметром **key** не существует ни одной очереди сообщений и в поле **msgflg** включен флаг **IPC_CREAT**. Поля **IPC_CREAT** и **IPC_EXCL** в **msgflg** играют ту же роль для очередей сообщений, что и **O_CREAT** и **O_EXCL** в параметре **mode** системной функции **open(2)**: функция **msgget** вернет ошибку, если в **msgflg** включены оба флага: **IPC_CREAT** и **IPC_EXCL**,- а такая очередь сообщений для **key** уже существует.
При создании очереди вспомогательные **9** битов параметра **msgflg** определяют права доступа к очереди сообщений. Эти биты прав имеют тот же формат и значение, что и параметр прав доступа для системных функций **open(2)** или **creat(2)**. (Права на исполнение не используются.)

# Системное программирование для современных платформ

Если создается новая очередь сообщений, то этот системный вызов инициализирует системную структуру данных в очереди сообщений **msqid_ds** следующим образом:

**msg_perm.cuid** и **msg_perm.uid** устанавливаются для эффективного идентификатора пользователя, запускающего вызывающий процесс.
**msg_perm.cgid** и **msg_perm.gid** устанавливаются для эффективного идентификатора группы вызывающего процесса.
Вспомогательные **9** битов **msg_perm.mode** приравниваются к вспомогательным **9**-и битам **msgflg**. Значение **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime** и **msg_rtime** равно нулю. **msg_ctime** устанавливается согласно текущему времени. **msg_qbytes** устанавливается согласно системному лимиту, заданному **MSGMNB**.
Если очередь сообщений уже существует, то проверяются права доступа к ней и производится подключение к очереди.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При удачном завершении возвращается идентификатор очереди. При ошибке возвращается **-1**, а переменной **errno** присваивается номер ошибки.

# Системное программирование для современных платформ

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

int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtyp, int
msgflg);
```

**ОПИСАНИЕ**

Для того чтобы отправить или получить сообщения, вызывающий процесс создает **struct msgbuf**. Поле **mtext** является массивом (или другой структурой), размер которого определяется неотрицательным **msgsz**. Сообщения нулевой длины (т.е. без поля **mtext**) разрешены. Поле **mtype** должно принимать только положительные целые значения, используемые процессом-получателем для выбора сообщений (см. раздел, касающийся **msgrcv**). Вызывающий процесс должен иметь права на запись для отправки данных, а при получении - права на чтение для этой очереди.

Системный вызов **msgsnd** добавляет сообщение, указанное параметром **msgp**, в очередь сообщений, идентификатор которой указывается параметром **msqid**.

Если для очереди доступно достаточно места, то **msgsnd** немедленно сообщает о нормальном завершении работы. (Емкость очереди определяется полем **msg_bytes** в ассоциированной структуре данных для очереди сообщений. Во время создания очереди это поле инициализируется в **MSGMNB** байтов, но это ограничение может быть изменено, используя **msgctl**.) Если для очереди недостаточно пространства, то поведением по умолчанию для **msgsnd** будет блокировка, до тех пор, пока не станет достаточно места. Если **IPC_NOWAIT** установлено в **msgflg** то вызов вместо этого выдаст ошибку **EAGAIN**.

Заблокированный вызов msgsnd также может выдать ошибку, если очередь удалена (в этом случае системный вызов выдаст ошибку, определив **errno** в значение **EAGAIN**.

**msgsnd** и **msgrcv** никогда автоматически не перезапускаются после прерывания обработчиком сигнала, независимо от установки флага **SA_RESTART** при установки обработчика сигнала.

До успешного завершения очереди сообщений структура данных будет обновляться следующим образом:

**msg_lspid** будет установлено равным идентификатору вызывающего процесса;
**msg_qnum** будет увеличено на 1;
**msg_stime** будет установлено равным текущему времени.

# Системное программирование для современных платформ

Системный вызов **msgrcv** записывает сообщение из очереди (идентификатор которой указан в **msqid**) в буфер **msgbuf** (находящийся в **msgp**), удаляя сообщение из очереди. Параметр **msgsz** задает максимальный размер элемента **mtext** структуры, находящейся по адресу, указанному в аргументе **msgp**. Если длина текста сообщения больше, чем **msgsz**, и флаг **msgflg** установлен в **MSG_NOERROR**, то текст сообщения будет урезан (а урезанная часть потеряна), иначе сообщение не удаляется из очереди, а системный вызов возвращает ошибку, установив значение **errno**, равное **E2BIG**.

Параметр **msgtyp** задает тип сообщения следующим образом:

если **msgtyp** равен нулю, то используется первое сообщение из очереди;

если **msgtyp** больше нуля, то из очереди берется первое сообщение типа **msgtyp** (если только во флаге **msgflg** нет бита **MSG_EXCEPT**. В этом случае из очереди берется первое сообщение, тип которого не равен **msgtyp**);

если **msgtyp** меньше нуля, то из очереди берется первое сообщение со значением, меньшим, чем абсолютное значение **msgtyp**.

Параметр **msgflg** состоит из комбинации следующих флагов:

**IPC_NOWAIT**, который указывает на немедленный возврат из функции, если в очереди нет сообщений необходимого типа. При этом системный вызов возвращает ошибку, устанавливая значение **errno** равным **ENOMSG**;

**MSG_EXCEPT**, который используется (если **msgtyp** больше, чем **0**) для чтения первого сообщения, тип которого не равен **msgtyp**;

**MSG_NOERROR**, используемый для урезания текста сообщения, размер которого больше **msgsz** байтов.

Если в очереди нет сообщения необходимого типа, и в **msgflg** не установлен флаг **IPC_NOWAIT**, то вызывающий процесс будет заблокирован до тех пор, пока не произойдет одно из следующих событий:

сообщение необходимого типа помещено в очередь;

очередь сообщений удалена из системы (в этом случае системный вызов возвращает ошибку, устанавливая значение **errno**, равное **EIDRM**).

Вызывающий процесс получает сигнал, который он должен обработать. В этом случае системный вызов возвращает ошибку, устанавливая значение переменной **errno**, равное **EINTR**. При удачном завершении операции структура данных очереди сообщений будет обновлена следующим образом:

**msg_lrpid** будет обозначать идентификатор вызывающего процесса;

**msg_qnum** уменьшается на **1**;

**msg_rtime** будет равно значению текущего времени.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При ошибке обе функции возвращают **-1**, а переменная **errno** приобретает соответствующее значение. В противном случае **msgsnd** возвращает **0**, а **msgrvc** возвращает количество байтов, скопированных в массив **mtext**.

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

**ОПИСАНИЕ**

Эта функция выполняет операцию, заданную в **cmd**, над очередью сообщений **msqid**. Возможные значения **cmd**:
**IPC_STAT -** скопировать информацию из структуры данных очереди сообщений, ассоциированных с **msqid** в структуру с адресом **buf** (у вызывающего должны быть права на чтение очереди сообщений).
**IPC_SET -** Записать значения некоторых элементов структуры **msqid_ds**, адрес которой указан в **buf**, в структуру данных из очереди сообщений, обновляя при этом его поле **msg_ctime**. Обновлены могут быть следующие поля: **msg_perm.uid**, **msg_perm.gid**, **msg_perm.mode** /* только неосновные 9 битов */, **msg_qbytes**.
Вызывающий процесс должен иметь соответствующие права или его действующий идентификатор пользователя должен соответствовать создателю (**msg_perm.cuid**) или владельцу (**msg_perm.uid**) очереди сообщений. Права суперпользователя требуются для установки значения **msg_qbytes** больше, чем **MSGMNB**.
**IPC_RMID -** Немедленно удалить очередь сообщений и связанную с ним структуру данных, "разбудив" все процессы, ожидающие записи или чтения этой очереди (при этом функция возвращает ошибку, а переменная **errno** приобретает значение **EIDRM**). Вызывающий процесс должен иметь соответствующие права или его действующий идентификатор пользователя должен соответствовать создателю или владельцу очереди сообщений.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**
При удачном завершении вызова возвращаемое значение равно нулю. При ошибке возвращается **-1**, а переменной errno присваивается номер ошибки.

# Системное программирование для современных платформ

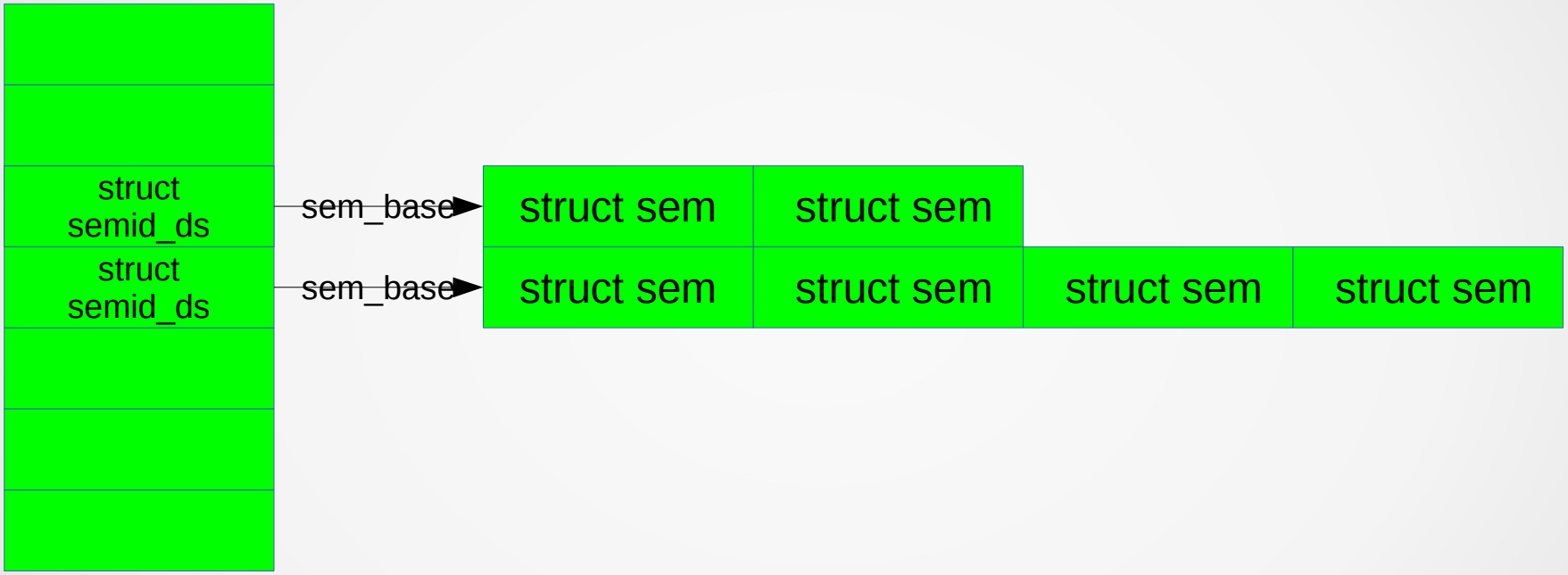
## Семафоры SysV IPC

# Системное программирование для современных платформ

Схема применения:

1. Процесс A, которому требуется выполнение определённых действий от процесса B, уменьшает значение одного или нескольких семафоров, после чего блокируется ядром
2. Когда процесс B готов позволить процессу A возобновить управление, он увеличивает значение ранее изменённых семафоров, так чтобы они стали больше 0

# Системное программирование для современных платформ



# Системное программирование для современных платформ

## Системные переменные

| | | |
|---|---|---|
| SEMMNI | Максимальное число наборов семафоров в системе в любой момент времени | блокировка |
| SEMMNS | Максимальное число семафоров в системе в любой момент времени | блокировка |
| SEMMSL | Максимальное число семафоров в одном наборе | ошибка |
| SEMOPM | Максимальное число семафоров в наборе, над которыми можно проводить операции одновременно | ошибка |

## API

| | |
|---|---|
| semget | Открытие или создание набора семафоров и получение её дескриптора |
| semop | Запрос и изменение значений семафоров |
| msgctl | Управление параметрами и закрытие набора семафоров |

# Системное программирование для современных платформ

Структуры данных

```
struct sembuf {
unsigned short sem_num;                    /* semaphore number */
short sem_op;                              /* semaphore operation */
short sem_flg;};                           /* operation flags */

struct ipc_perm {
  key_t  key;
  ushort uid;                              /* owner euid and egid */
  ushort gid;
  ushort cuid;                             /* creator euid and egid */
  ushort cgid;
  ushort mode;                             /* access modes */
  ushort seq;};                            /* slot usage sequence number */
```

# Системное программирование для современных платформ

```
struct sem {
short   sempid;                                  /* pid of last operation */
ushort  semval;                                  /* current value */
ushort  semncnt;                  /* num procs awaiting increase in semval */
ushort  semzcnt;};                        /* num procs awaiting semval = 0 */

struct semid_ds {
struct ipc_perm sem_perm;                                /* permissions */
time_t sem_otime;                                    /* last semop time */
time_t sem_ctime;                                   /* last change time */
struct sem *sem_base;               /* pointer to first semaphore in array */
struct wait_queue *eventn;
struct wait_queue *eventz;
struct sem_undo  *undo;                          /* undo requests on this array */
ushort sem_nsems;};                      /* nombers of semaphores in array */
```

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

**ОПИСАНИЕ**

Системный вызов возвращает идентификатор набора семафоров, связанный с аргументом **key**. Создается новый набор из семафоров **nsems**, если значение **key** равно **IPC_PRIVATE** или с ключом **key** не связано ни одного существующего набора семафора, а флаг IPC_CREAT установлен. Наличие в **semflg** полей **IPC_CREAT** и **IPC_EXCL** играет ту же роль, что и наличие **O_CREAT** и **O_EXCL** в аргументе, заданном для режима системного вызова **open(2)**. Например, функция **semget** не выполняется, если **semflg** имеет флаги **IPC_CREAT** и **IPC_EXCL**, а набор семафоров уже существует для **key**.

Младшие **9** битов аргумента **semflg** определяют права доступа (для владельца, группы и остальных) к набору семафоров. Эти биты имеют тот же формат и те же значения, что и аргумент **mode** в функциях **open(2)** или **creat(2)** (хотя права на исполнение игнорируются, а права на запись приравниваются к правам на изменение значений семафора).

# Системное программирование для современных платформ

При создании нового набора семафоров **semget** инициализирует связанную с семафором структуру данных **semid_ds** следующим образом:

**sem_perm.cuid** и **sem_perm.uid** присваиваются значения идентификаторов эффективного пользователя вызывающего процесса;
**sem_perm.cgid** и **sem_perm.gid** присваиваются значения идентификаторов эффективной группы пользователей вызывающего процесса;
Младшим **9** битам **sem_perm.mode** присваивается значение младших **9** битов **semflg**;
**sem_nsems** присваивается значение **nsems**;
**sem_otime** присваивается значение **0**.
**sem_ctime** устанавливается на текущее время.

Аргумент **nsems** может быть равен **0**, если набор семафоров не создается. Иначе аргумент **nsems** должен быть больше, чем **0** и меньше, либо равен максимальному количеству семафоров в наборе (**SEMMSL**).

Если набор семафоров уже существует, то проверяются права доступа.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При удачном завершении возвращаемое значение будет представлять собой идентификатор набора семафоров (целое неотрицательное значение), иначе возвращается **-1**, а переменной **errno** присваивается номер ошибки.

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);

int semtimedop(int semid, struct sembuf *sops, unsigned nsops, struct timespec
*timeout);
```

**ОПИСАНИЕ**

Функция производит операции над выбранными элементами из набора семафоров **semid**. Каждый из элементов **nsops** в массиве **sops** определяет операцию, производимую над семафором в структуре **struct sembuf**. Флаги в **sem_flg** могут иметь значения **IPC_NOWAIT** и **SEM_UNDO**. Если стоит флаг **SEM_UNDO**, то будет выполнена обратная операция при закрытии процесса. Обратная операция будет выполнена только при условии выполнения предыдущей операции. Каждая операция выполняется над семафором **sem_num** набора, где первый семафор набора имеет номер **0**, и его значение равно одному из трех следующих:

- Если **sem_op** является положительным целым числом, то оно добавляется к **semval**. Если для операции стоит флаг **SEM_UNDO**, то система изменяет счетчик обратных операций для этого семафора. Обратная операция осуществляется всегда и поэтому никогда не может перейти в режим ожидания. Вызывающий процесс должен иметь права на изменение набора.
- Если **sem_op** равен нулю, то процесс должен иметь права на чтение набора. Если **semval** равен нулю, то операция производится без наличия дополнительных прав. Иначе, если флаг **IPC_NOWAIT** стоит в поле семафора **sem_flg**, системный вызов не выполняется (производятся все обратные операции), а **errno** присваивается значение **EAGAIN**. В другом случае **semzcnt** увеличивается на единицу, а процесс переходит в режим ожидания до того момента, когда

* **semval** становится равным **0**, а значение **semzcnt** увеличивается.
* Набор семафоров удален, системный вызов не выполняется, а **errno** присваивается значение **EIDRM**.
* Вызывающий процесс получает сигнал, который должен быть обработан; когда это происходит, значение **semzcnt** увеличивается, а системный вызов не выполняется, а переменной **errno** присваивается значение **EINTR**.
* Лимит времени, определенный timeout в вызове **semtimedop** истек: системный вызов выдает ошибку, а переменная **errno** устанавливается в **EAGAIN**.

# Системное программирование для современных платформ

- Если **sem_op** меньше нуля, то процесс должен иметь права на изменение набора. Если **semval** больше или равен абсолютному значению **sem_op**, то абсолютное значение **sem_op** приравнивается к **semval**. Если для этой операции установлен флаг **SEM_UNDO**, то счетчик обратных операций для этого семафора обновляется, а выполнение операции продолжается. Иначе, если флаг **IPC_NOWAIT** стоит в поле **sem_flg**, системный вызов не выполняется (производятся все обратные операции), а **errno** присваивается значение **EAGAIN**. В ином случае **semncnt** увеличивается на единицу, а процесс переходит в режим ожидания до того момента, когда

* **semval** становится больше или равно абсолютному значению **sem_op**, значение **semncnt** уменьшается, абсолютное значение **sem_op** приравнивается к **semval**, и если флаг **SEM_UNDO** установлен для этой операции, то система обновляет счетчик обратных операций для этого семафора.
* При удалении набора семафоров системный вызов не выполняется, а переменной **errno** присваивается значение **EIDRM**.
* Вызывающий процесс получает сигнал, после чего **semncnt** увеличивается, системный вызов не выполняется, а переменной **errno** присваивается значение **EINTR**.
* Лимит времени, определенный **timeout** в вызове **semtimedop** истек: системный вызов выдает ошибку, а переменная **errno** устанавливается в **EAGAIN**.

В случае выполнения **sempid**, являющейся членом структуры, **sem** приобретает значение идентификатора процесса. Так или иначе, **sem_otime** и **sem_ctime** устанавливаются на текущее время.

Функция **semtimedop** ведет себя идентично функции **semop** за исключением того, что в случаях, когда вызывающий процесс будет спать, длительность этого сна ограничена количеством времени, определенного структурой **timespec** чей адрес передается в параметре **timeout**. Если достигнут указанный лимит времени, то системный вызов выдает ошибку, а **errno** устанавливается в **EAGAIN** (и ни одна из операций в **sops** не выполняется). Если параметр **timeout** равен **NULL**, то **semtimedop** ведет себя точно так же, как и **semop**.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При успешном выполнении системного вызова возвращаемое значение равно нулю, в случае ошибки оно равно **-1**, а переменной **errno** присваивается номер ошибки.

# Системное программирование для современных платформ

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

int semctl(int semid, int semnum, int cmd, ...);
```

**ОПИСАНИЕ**

Функция **semctl** позволяет выполнять операции, определенные в **cmd** над набором семафоров, указанным в semid или над семафором с номером **semnum** из этого набора. (Семафоры нумеруются, начиная с 0.)
Функция имеет три или четыре аргумента. Если аргументов четыре, то вызов выглядит как **semctl(semid,semnum,cmd,arg);** где четвертый аргумент **arg** имеет тип **union semun,** определенный как

```
union semun {
      int val;                  /* value for SETVAL */
      struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
      unsigned short *array;    /* array for GETALL, SETALL */
                                /* Linux specific part: */
      struct seminfo *__buf;    /* buffer for IPC_INFO */
};
```

# Системное программирование для современных платформ

Аргумент **cmd** может принимать следующие значения:

**IPC_STAT -** скопировать информацию из структуры данных набора семафоров в структуру, указанную в **arg.buf**. Аргумент **semnum** игнорируется. Вызывающий процесс должен прочитать привилегии доступа в наборе семафоров.
**IPC_SET -** изменить значения некоторых членов структуры **semid_ds**, на которую указывает **arg.buf**, в структуру данных набора семафоров и обновите **sem_ctime**. Присвоить следующим полям структуры данных **struct semid_ds** соответствующие значения, на которые указывает **arg.buf**, **sem_perm.uid**, **sem_perm.gid**, **sem_perm.mode** /* Только младшие 9 битов */
Эта команда может выполняться только процессом, который имеет действующий идентификатор пользователя, равный либо идентификатору суперпользователя, либо создателя или владельца набора семафоров. Аргумент **semnum** игнорируется.
**IPC_RMID -** Немедленно удалить из системы набор семафоров и структуры его данных, запускающие все процессы, находящиеся в режиме ожидания (при этом возвращается сообщение об ошибке, а errno присваивается значение **EIDRM**). Эта команда может выполняться только процессом, который имеет действующий идентификатор пользователя, равный либо идентификатору суперпользователя, либо создателя или владельца набора семафоров. Аргумент **semnum** игнорируется.
**GETALL -** Возвращает значение **semval** всем семафорам в массиве **arg.array**. Аргумент **semnum** игнорируется. Для этого вызывающему процессу нужны права на чтение.
**GETNCNT -** Системный вызов возвращает значение **semncnt** семафору **semnum-th** (например, число процессов, ожидающих увеличения значения **semval** семафора **semnum-th**). Для этого вызывающему процессу нужны права на чтение.

# Системное программирование для современных платформ

**GETPID -** Системный вызов возвращает значение **sempid** семафору **semnum-th** (например, идентификатор процесса, который последним делал вызов semop семафору **semnum-th**). Для этого вызывающему процессу нужны права на чтение.
**GETVAL -** системный вызов возвращает значение **semval** семафору **semnum-th**. Для этого вызывающему процессу нужны права на чтение.
**GETZCNT -** Системный вызов возвращает значение **semzcnt** семафору **semnum-th** (например, количество процессов, ожидающих, чтобы значение **semval** семафора **semnum-th** стало равным нулю). Для этого вызывающему процессу нужны права на чтение.
**SETALL -** Установить значение **semval** всех семафоров равным значениям элементов массива, на который указывает **arg.array**, изменяя также **sem_ctime**, являющееся членом структуры **semid_ds**; а эта структура ассоциируется с набором семафоров. История отменяемых операций удаляется для всех измененных семафоров во всех процессах. Процессы, находящиеся в очереди, активизируются, если **semval** становится равным нулю или значение его увеличивается. Аргумент semnum игнорируется. Для этого вызывающему процессу нужны права на чтение.
**SETVAL -** Установите значение **semval** на указанное в **arg.val** для всех семафоров **semnum-th**, изменяя также **sem_ctime** в структуре **semid_ds**, соотносимой с набором семафоров. История отмененных операций удаляется для всех измененных семафоров во всех процессах. Процессы, находящиеся в очереди, активизируются, если **semval** становится равным нулю или значение его увеличивается. Вызывающему процессу потребуется право на его изменение.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При ошибке **semctl** вернет **-1**, а переменной errno присваивается номер ошибки. Иначе говоря, системный вызов возвращает положительное значение, зависящее от **cmd:**
**GETNCNT** значение семафора равно **semncnt.**
**GETPID** значение семафора равно **sempid.**
**GETVAL** значение семафора равно **semval.**
**GETZCNT** значение семафора равно **semzcnt.**

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <string.h> /* For strerror(3c) */
#include <errno.h> /* For errno */
#include <unistd.h> /* rand(3c) */
#include <stdio.h>

int main (int argc, char **argv) {

        key_t ipckey;
        int semid;
        struct sembuf sem[2]; /* sembuf defined in sys/sem.h */

        ipckey = ftok("/tmp/foo", 42); /* Generate the ipc key */

        semid = semget(ipckey, 1, 0666 | IPC_CREAT); /* Set up the semaphore set. 4 == READ, 2 == ALTER */
        if (semid < 0) {
                printf("Error - %s\n", strerror(errno));
                _exit(1);}

        /* These never change so leave them outside the loop */
        sem[0].sem_num = 0;
        sem[1].sem_num = 0;
        sem[0].sem_flg = SEM_UNDO; /* Release semaphore on exit */
        sem[1].sem_flg = SEM_UNDO; /* Release semaphore on exit */
        while(1) { /* loop forever */
                printf("[%s] Waiting for the semaphore to be released\n", argv[1]);
                /* Set up two semaphore operations */
                sem[0].sem_op = 0; /* Wait for zero */
                sem[1].sem_op = 1; /* Add 1 to lock it*/
                semop(semid, sem, 2);
                printf("[%s] I have the semaphore\n", argv[1]);

                sleep(rand() % 3); /* Critical section, sleep for 0-2 seconds */

                sem[0].sem_op = -1; /* Decrement to unlock */
                semop(semid, sem, 1);
                printf("[%s] Released semaphore\n", argv[1]);

                sleep(rand() % 3); /* Sleep 0-2 seconds */
        }

}
```

# Системное программирование для современных платформ

## Области разделяемой памяти SysV IPC

# Системное программирование для современных платформ

## Системные переменные

| | | |
|---|---|---|
| SHMMNI | Максимальное число разделяемых областей памяти в системе в любой момент времени | ошибка |
| SHMMIN | Минимальный раздел разделяемой области памяти(в байтах) | ошибка |
| SHMMAX | Максимальный раздел разделяемой области памяти(в байтах) | блокировка |

## API

| | |
|---|---|
| shmget | Открытие или создание разделяемой и получение её дескриптора |
| shmat | Присоединить разделяемую область памяти к виртуальному адресному пространству процесса |
| shmdt | Отсоединить разделяемую область памяти от виртуального адресного пространства процесса |
| shmctl | Управление параметрами и закрытие очереди |

# Системное программирование для современных платформ

Структуры данных

```
struct shmid_ds {
    struct ipc_perm shm_perm;                                   /* права операции */
    int             shm_segsz;                          /* размер сегмента (в байтах) */
    time_t          shm_atime;                        /* время последнего подключения */
    time_t          shm_dtime;                         /* время последнего отключения */
    time_t          shm_ctime;                         /* время последнего изменения */
    unsigned short  shm_cpid;                    /* идентификатор процесса создателя */
    unsigned short  shm_lpid;                /* идентификатор последнего пользователя */
    short           shm_nattch;};                          /* количество подключений */

struct ipc_perm {
    key_t  key;
    ushort uid;       /* действующие идентификаторы владельца и группы euid и egid */
    ushort gid;
    ushort cuid;                /* действующие идентификаторы создателя euid и egid */
    ushort cgid;
    ushort mode;                                              /* младшие 9 битов shmflg */
    ushort seq;};                                            /* номер последовательности */
```

# Системное программирование для современных платформ

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

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

**ОПИСАНИЕ**

**shmget()** возвращает идентификатор разделяемому сегменту памяти, соответствующий значению аргумента **key**. Создается новый разделяемый сегмент памяти с размером **size**, выровненный по размеру страниц, если значение **key** равно **IPC_PRIVATE** или если значение **key** не равно **IPC_PRIVATE** и нет идентификатора, соответствующего **key**; причем, установлен флаг **IPC_CREAT**. Поле **shmflg** состоит из:

**IPC_CREAT -** служит для создания нового сегмента. Если этого флага нет, то функция **shmget()** будет искать сегмент, соответствующий ключу **key** и затем проверит, имеет ли пользователь права на доступ к сегменту.
**IPC_EXCL -** используется совместно с **IPC_CREAT** для того, чтобы не создавать существующий сегмент заново.
**mode_flags** (младшие 9 битов) - указывают на права хозяина, группы и др. В некоторых системах не используются.
Если создается новый сегмент, то права доступа копируются из **shmflg** в **shm_perm**, являющийся членом структуры **shmid_ds**, которая определяет сегмент.

При создании нового сегмента системный вызов инициализирует структуру данных **shmid_ds** следующим образом:
**shm_perm.cuid** и **shm_perm.uid** становятся равными значению идентификатора эффективного пользователя вызывающего процесса.
**shm_perm.cgid** и **shm_perm.gid** устанавливаются равными идентификатору эффективной группы пользователей вызывающего процесса.
Младшим 9-и битам **shm_perm.mode** присваивается значение младших 9-и битов **shmflg**.
**shm_segsz** присваивается значение **size**. Устанавливаемое значение
**shm_lpid**, **shm_nattch**, **shm_atime** и **shm_dtime** становится равным нулю.
**shm_ctime** устанавливается на текущее время.
Если сегмент уже существет, то права доступа подтверждаются, а проверка производится для того, чтобы убедиться, что сегмент не помечен на удаление.

**СИСТЕМНЫЕ ВЫЗОВЫ**
**fork()** - После выполнения команды **fork()** дочерний процесс наследует подключаемые к нему разделяемые сегменты памяти.
**exec()** - После выполнения команды **exec()** все подключаемые к процессу разделяемые сегменты памяти отключаются от него, но не удаляются.
**exit()** - По завершении **exit()** все подключенные разделяемые сегменты памяти отключаются (но не удаляются).

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**
При удачном завершении вызова возвращается идентификатор сегмента **shmid**, и **-1** при ошибке.

# Системное программирование для современных платформ

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);

int shmdt(const void *shmaddr);
```

**ОПИСАНИЕ**

Функция **shmat** подстыковывает сегмент разделяемой памяти **shmid** к адресному пространству вызывающего процесса. Адрес подстыковываемого сегмента определяется **shmaddr** с помощью одного из перечисленных ниже критериев:

* Если **shmaddr** равен **NULL**, то система выбирает для подстыкованного сегмента подходящий (неиспользованный) адрес.
* Если **shmaddr** не равен **NULL**, а в поле **shmflg** включен флаг **SHM_RND**, то подстыковка производится по адресу **shmaddr**, округленному вниз до ближайшего кратного **SHMLBA**. В противном случае **shmaddr** должен быть округленным до размера страницы адресом, к которому производится подстыковка.

Если в поле **shmflg** включен флаг **SHM_RDONLY**, то подстыковываемый сегмент будет доступен только для чтения, и вызывающий процесс должен иметь права на чтение этого сегмента. Иначе, сегмент будет доступен для чтения и записи, и у процесса должны быть соответствующие права. Сегментов "только-запись" не существует.

Флаг **SHM_REMAP** (специфичный для **Linux**) может быть указан в **shmflg** для обозначения того, что распределение сегмента должно замещать любые существующие распределения в диапазоне, начиная с **shmaddr** и до размера сегмента. (Обычно выдается ошибка **EINVAL** если уже существует распределение в этом диапазоне адресов.) В этом случае **shmaddr** не должно быть равно **NULL**.

Значение **brk** вызывающего процесса подстыковкой не изменяется. При завершении работы процесса сегмент будет отстыкован. Один и тот же сегмент может быть подстыкован в адресное пространство процесса несколько раз, как "только для чтения", так и в режиме "чтение-запись".

# Системное программирование для современных платформ

При удачном выполнении системный вызов **shmat** обновляет содержимое структуры **shmid_ds**, связанной с разделяемым сегментом памяти, следующим образом:

**shm_atime** устанавливается в текущее время.
**shm_lpid** устанавливается в идентификатор вызывающего процесса.
**shm_nattch** увеличивается на **1**.
Заметьте, что пристыковка производится и в том случае, если пристыковываемый сегмент помечен на удаление.

Функция **shmdt** отстыковывает сегмент разделяемой памяти, находящийся по адресу **shmaddr**, от адресного пространства вызвающего процесса. Отстыковываемый сегмент должен быть среди пристыкованных ранее функцией **shmat**. Параметр **shmaddr** должен быть равен значению, которое возвратила соответствующая функция **shmat**.

При удачном выполнении системный вызов **shmdt** обновляет содержимое структуры **shmid_ds**, связанной с разделяемым сегментом памяти, следующим образом:

**shm_dtime** устанавливается в текущее время.
**shm_lpid** устанавливается в идентификатор вызывающего процесса.
**shm_nattch** уменьшается на **1**. Если это значение становится равным **0**, а сегмент помечен на удаление, то сегмент удаляется из памяти. Эта функция освобождает занятую ранее этим сегментом область памяти в адресном пространстве процесса.

**СИСТЕМНЫЕ ВЫЗОВЫ**
**fork() -** При исполнении **fork()** дочернии процесс наследует пристыкованные сегменты разделяемой памяти.
**exec() -** При исполнении **exec()** все подстыкованные сегменты памяти отстыковываются от процесса.
**exit() -** При исполнении **exit()** все подстыкованные сегменты памяти отстыковываются от процесса.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**
При ошибке обе функции возвращают **-1**, а переменной **errno** присваивается номер ошибки. При удачном выполнении **shmat** возвращает адрес подстыкованного сегмента памяти, а **shmdt** возвращает **0**.

# Системное программирование для современных платформ

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

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

**ОПИСАНИЕ**

**shmctl()** позволяет пользователю получать информацию о разделяемых сегментах памяти, устанавливать владельца, группу разделяемого сегмента, права на него; эта функция может также удалить сегмент. Информация о сегменте, которая находится в **shmid**, возвращается в структуру **shmid_ds** :

Значения аргумента **cmds** могут быть следующими:
**IPC_STAT -** используется для копирования информации о сегменте в буфер **buf**. Пользователь должен иметь права на чтение сегмента **read**.
**IPC_SET -** используется для применения пользовательских изменений к содержимому полей **uid**, **gid** или **mode** в структуре **shm_perms**. Используются только младшие 9 битов **mode**. Поле **shm_ctime** тоже обновляется. Пользователь должен быть владельцем, создателем процесса или суперпользователем.
**IPC_RMID -** используется для пометки сегмента как удаленного. Сегмент будет удален после отключения (например, когда поле **shm_nattch** ассоциированной структуры **shmid_ds** равно нулю). Пользователь должен быть владельцем, создателем или суперпользователем процесса. Пользователь должен удостовериться, что сегмент удален; иначе страницы, которые не были удалены, останутся в памяти или в разделе подкачки.

Также процессы с соответствующими привилегиями могут предотвратить или разрешить подкачку разделяемого сегмента памяти при помощи следующих команд **cmds** (применимо только в ОС, позволяющих управлять флагом **SWAPLCK** страницы):
**SHM_LOCK -** запретить подкачку разделяемого сегмента памяти. После блокировки страницы должны находиться в памяти.
**SHM_UNLOCK -** разрешить подкачку сегмента.
Процессы, которым разрешено использовать **SHM_LOCK** и **SHM_UNLOCK** при запуске их с возможностью **CAP_IPC_LOCK** (обычно выдаваемой только для **root**) или если их текущий лимит ресурсов **RLIMIT_MEMLOCK** не равен нулю.

# Системное программирование для современных платформ

Вызовы **IPC_INFO**, **SHM_STAT** и **SHM_INFO** используются программой **ipcs(8)** для получения информации о выделенных ресурсах. В будущем они могут быть по необходимости изменены или вынесены в файловую систему **proc**.

**СИСТЕМНЫЕ ВЫЗОВЫ**

**fork() -** При исполнении **fork()** дочернии процесс наследует пристыкованные сегменты разделяемой памяти.
**exec() -** При исполнении **exec()** все подстыкованные сегменты памяти отстыковываются от процесса.
**exit() -** При исполнении **exit()** все подстыкованные сегменты памяти отстыковываются от процесса.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ**

При удачном выполнении возвращается **0**, а при ошибке **-1**.

# Системное программирование для современных платформ

**Утилиты**

**ipcs [ -asmq ] [ -tclup ]**
**ipcs [ -smq ] -i id**
**ipcs -h**

**ОПИСАНИЕ**

**ipcs** выводит информацию о возможностях **IPC,** к которым вызывающий процесс имеет доступ по чтению.

Опция -i позволяет отдельно указать дескриптор ресурса id. В этом случаем будет выведена информация только по указанному ресурсу.

Можно указывать следующие типы ресурсов:
**-m** сегменты разделяемой памяти
**-q** очереди сообщений
**-s** массивы семафоров
**-a** все вместе (по умолчанию)

Формат вывода может быть задан следующим образом:
**-t** время
**-p** идентификатор процесса **(pid)**
**-c** создатель ресурса
**-l** пределы
**-u** итоговые данные

# Системное программирование для современных платформ

**ipcrm - удалить ресурс IPC**

**СИНТАКСИС**

**ipcrm [ shm | msg | sem ] id**

**ОПИСАНИЕ**

**ipcrm** удаляет ресурс **IPC,** указанный его дескриптором (**id**).

# Системное программирование для современных платформ

Спасибо за внимание ;)