**Лабораторная работа №4**

**Средства System V IPC. Организация работы с разделяемой памятью в UNIX.**

***Цель:*** *Ознакомиться со средствами передачи информации System V IPC; организовать взаимодействие двух программ через разделяемую память.*

## 1. Теоретический блок

### Преимущества и недостатки потокового обмена данными

Потоковые механизмы достаточно просты в реализации и удобны для использования, но имеют ряд существенных недостатков:

• Операции чтения и записи не анализируют содержимое передаваемых данных. Процесс, прочитавший 20 байт из потока, не может сказать, были ли они записаны одним процессом или несколькими, записывались ли они за один раз или было, например, выполнено 4 операции записи по 5 байт. Данные в потоке никак не интерпретируются системой. Если требуется какая-либо интерпретация данных, то передающий и принимающий процессы должны заранее согласовать свои действия и уметь осуществлять ее самостоятельно.

• Для передачи информации от одного процесса к другому требуется, как минимум, две операции копирования данных: первый раз — из адресного пространства передающего процесса в системный буфер, второй раз — из системного буфера в адресное пространство принимающего процесса.

• Процессы, обменивающиеся информацией, должны одновременно существовать в вычислительной системе. Нельзя записать информацию в поток с помощью одного процесса, завершить его, а затем, через некоторое время, запустить другой процесс и прочитать записанную информацию.

### Понятие о System V IPC

Указанные выше недостатки потоков данных привели к разработке других механизмов передачи информации между процессами. Часть этих механизмов, впервые появившихся в UNIX System V и впоследствии перекочевавших оттуда

практически во все современные версии операционной системы UNIX, получила общее название *System V IPC* (IPC — сокращение от Interprocess Communications). В группу System V IPC входят: очереди сообщений, разделяемая память и семафоры. Эти средства организации взаимодействия процессов связаны не только общностью происхождения, но и обладают схожим интерфейсом для выполнения подобных операций, например, для выделения и освобождения соответствующего ресурса в системе. Мы будем рассматривать их в порядке от менее семантически нагруженных с точки зрения операционной системы к более семантически нагруженным. Иными словами, чем позже мы начнем заниматься каким-либо механизмом из System V IPC, тем больше действий по интерпретации передаваемой информации придется выполнять операционной системе при использовании этого механизма.

**Пространство имен. Адресация в System V IPC. Функция ftok()**

Все средства связи из System V IPC, как и уже рассмотренные нами pipe и FIFO, являются средствами связи с непрямой адресацией. Как мы установили на предыдущем семинаре, для организации взаимодействия неродственных процессов с помощью средства связи с непрямой адресацией необходимо, чтобы это средство связи имело имя. Отсутствие имен у pip'ов позволяет процессам получать информацию о расположении pip'а в системе и его состоянии только через родственные связи. Наличие ассоциированного имени у FIFO — имени специализированного файла в файловой системе — позволяет неродственным процессам получать эту информацию через интерфейс файловой системы.

Множество всех возможных имен для объектов какого-либо вида принято называть пространством имен соответствующего вида объектов. Для FIFO, пространством имен является множество всех допустимых имен файлов в файловой системе. Для всех объектов из System V IPC таким пространством имен является множество значений некоторого целочисленного типа данных — key_t - ключа. Причем программисту не позволено напрямую присваивать значение ключа, это значение задается опосредованно: через комбинацию имени какого-

либо файла, уже существующего в файловой системе, и небольшого целого числа — например, номера экземпляра средства связи.

Такой хитрый способ получения значения ключа связан с двумя соображениями:

• Если разрешить программистам самим присваивать значение ключа для идентификации средств связи, то не исключено, что два программиста случайно воспользуются одним и тем же значением, не подозревая об этом. Тогда их процессы будут несанкционированно взаимодействовать через одно и то же средство коммуникации, что может привести к нестандартному поведению этих процессов. Поэтому основным компонентом значения ключа является преобразованное в числовое значение полное имя некоторого файла, доступ к которому на чтение разрешен процессу. Каждый программист имеет возможность использовать для этой цели свой специфический файл, например исполняемый файл, связанный с одним из взаимодействующих процессов. Следует отметить, что преобразование из текстового имени файла в число основывается на расположении указанного файла на жестком диске или ином физическом носителе. Поэтому для образования ключа следует применять файлы, не меняющие своего положения в течение времени организации взаимодействия процессов.

• Второй компонент значения ключа используется для того, чтобы позволить программисту связать с одним и тем же именем файла более одного экземпляра каждого средства связи. В качестве такого компонента можно задавать порядковый номер соответствующего экземпляра.

Получение значения ключа из двух компонентов осуществляется функцией *ftok()*.

Прототип функции

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

Описание функции

Функция *ftok* служит для преобразования имени существующего файла и небольшого целого числа, например, порядкового номера экземпляра средств связи, в ключ System V IPC.

Параметр *pathname* должен являться указателем на имя существующего файла, доступного для процесса, вызывающего функцию.

Параметр *proj* - это небольшое целое число, характеризующее экземпляр средства связи.

В случае невозможности генерации ключа функция возвращает отрицательное значение, в противном случае она возвращает значение сгенерированного ключа. Тип данных *key_t* обычно представляет собой 32-битовое целое.

Еще раз подчеркнем три важных момента, связанных с использованием имени файла для получения ключа. Во-первых, необходимо указывать имя файла, который уже существует в файловой системе и для которого процесс имеет право доступа на чтение (не путайте с заданием имени файла при создании FIFO, где указывалось имя для вновь создаваемого специального файла). Во-вторых, указанный файл должен сохранять свое положение на диске до тех пор, пока все процессы, участвующие во взаимодействии, не получат ключ System V IPC. В-третьих, задание имени файла, как одного из компонентов для получения ключа, ни в коем случае не означает, что информация, передаваемая с помощью ассоциированного средства связи, будет располагаться в этом файле. Информация будет храниться внутри адресного пространства операционной системы, а заданное имя файла лишь позволяет различным процессам сгенерировать идентичные ключи.

**Дескрипторы System V IPC**

Информацию о потоках ввода-вывода, с которыми имеет дело текущий процесс, в частности о pip'ax и FIFO, операционная система хранит в таблице открытых файлов процесса. Системные вызовы, осуществляющие операции над потоком, используют в качестве параметра индекс элемента таблицы открытых

файлов, соответствующий потоку, — файловый дескриптор. Использование файловых дескрипторов для идентификации потоков внутри процесса позволяет применять к ним уже существующий интерфейс для работы с файлами, но в то же время приводит к автоматическому закрытию потоков при завершении процесса. Этим, в частности, объясняется один из перечисленных выше недостатков потоковой передачи информации.

При реализации компонентов System V IPC была принята другая концепция. Ядро операционной системы хранит информацию обо всех средствах System V IPC, используемых в системе, вне контекста пользовательских процессов. При создании нового средства связи или получении доступа к уже существующему процесс получает неотрицательное целое число — *дескриптор (идентификатор) этого средства связи,* который однозначно идентифицирует его во всей вычислительной системе. Этот дескриптор должен передаваться в качестве параметра всем системным вызовам, осуществляющим дальнейшие операции над соответствующим средством System V I PC.

Подобная концепция позволяет устранить один из самых существенных недостатков, присущих потоковым средствам связи — требование одновременного существования взаимодействующих процессов, но в то же время требует повышенной осторожности для того, чтобы процесс, получающий информацию, не принял взамен новых старые данные, случайно оставленные в механизме коммуникации.

**Разделяемая память в UNIX. Системные вызовы shmget(), shmat(), shmdt()**

С точки зрения операционной системы, наименее семантически нагруженным средством System V IPC является разделяемая память (shared memory). Для текущего семинара нам достаточно знать, что операционная система может позволить нескольким процессам совместно использовать некоторую область адресного пространства.

Все средства связи System V IPC требуют предварительных инициализирующих действий (создания) для организации взаимодействия процессов.

Для создания области разделяемой памяти с определенным ключом или доступа по ключу к уже существующей области применяется системный вызов *shmget()*. Существует два варианта его использования для создания новой области разделяемой памяти:

• Стандартный способ. В качестве значения ключа системному вызову поставляется значение, сформированное функцией *ftok()* для некоторого имени файла и номера экземпляра области разделяемой памяти. В качестве флагов поставляется комбинация прав доступа к создаваемому сегменту и флага *IPC_CREAT*. Если сегмент для данного ключа еще не существует, то система будет пытаться создать его с указанными правами доступа. Если же вдруг он уже существовал, то мы просто получим его дескриптор. Возможно добавление к этой комбинации флагов флага *IPC_EXCL*. Этот флаг гарантирует нормальное завершение системного вызова только в том случае, если сегмент действительно был создан (т. е. ранее он не существовал), если же сегмент существовал, то системный вызов завершится с ошибкой, и значение системной переменной *errno*, описанной в файле *errno. h*, будет установлено в *EEXIST*.

• Нестандартный способ. В качестве значения ключа указывается специальное значение *IPC_PRIVATE*. Использование значения *IPC_PRIVATE* всегда приводит к попытке создания нового сегмента разделяемой памяти с заданными правами доступа и с ключом, который не совпадает со значением ключа ни одного из уже существующих сегментов и который не может быть получен с помощью функции *ftok()* ни при одной комбинации ее параметров. Наличие флагов *IPC_CREAT* и *IPC_EXCL* в этом случае игнорируется.

Прототип системного вызова *shmget()*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg) ;
```

Описание системного вызова

Системный вызов *shmget* предназначен для выполнения операции доступа к сегменту разделяемой памяти и, в случае его успешного завершения, возвращает дескриптор System V IPC для этого сегмента (целое неотрицательное число, однозначно характеризующее сегмент внутри вычислительной системы и использующееся в дальнейшем для других операций с ним).

Параметр *key* является ключом System V IPC для сегмента, т. е. фактически его именем из пространства имен System V IPC. В качестве значения этого параметра может использоваться значение ключа, полученное с помощью функции *ftok()*, или специальное значение *IPC_PRIVATE*. Использование значения *IPC_PRIVATE* всегда приводит к попытке создания нового сегмента разделяемой памяти с ключом, который не совпадает со значением ключа ни одного из уже существующих сегментов и который не может быть получен с помощью функции *ftok()* ни при одной комбинации ее параметров.

Параметр *SIZE* определяет размер создаваемого или уже существующего сегмента в байтах. Если сегмент с указанным ключом уже существует, но его размер не совладает с указанным в параметре *SIZE*, констатируется возникновение ошибки.

Параметр *shmflg* - флаги - играет роль только при создании нового сегмента разделяемой памяти и определяет права различных пользователей при доступе к сегменту, а также необходимость создания нового сегмента и поведение системного вызова при попытке создания. Он является некоторой комбинацией (с помощью операции побитовое «или» («|») следующих предопределенных значений и восьмеричных порав доступа:

*IPC_CREAT* - если сегмента для указанного ключа не существует, он должен быть создан;

*IPC_EXCL* - применяется совместно с флагом *IPC_CREAT*. При совместном их использовании и существовании сегмента с указанным ключом, доступ к сегменту не производится и констатируется ошибочная ситуация, при этом переменная *errno*, описанная в файле <*ermo. h*>, примет значение *EEXIST*;

0400 - разрешено чтение для пользователя, создавшего сегмент;

0200 - разрешена запись для пользователя, создавшего сегмент;

0040 - разрешено чтение для группы пользователя, создавшего сегмент;

0020 - разрешена запись для группы пользователя, создавшего сегмент;

0004 - разрешено чтение для всех остальных пользователей;

0002 - разрешена запись для всех остальных пользователей.

Возвращаемое значение

Системный вызов возвращает значение дескриптора System V IPC для сегмента разделяемой памяти при нормальном завершении и значение -1 при возникновении ошибки.

Доступ к созданной области разделяемой памяти в дальнейшем обеспечивается ее дескриптором, который вернет системный вызов *shmget()*. Доступ к уже существующей области также может осуществляться двумя способами:

• Если мы знаем ее ключ, то, используя вызов *shmget()*, можем получить ее дескриптор. В этом случае нельзя указывать в качестве составной части флагов флаг *IPC_EXCL*, а значение ключа, естественно, не может быть *IPC_PRIVATE*. Права доступа игнорируются, а размер области должен совпадать с размером, указанным при ее создании.

• Либо мы можем воспользоваться тем, что дескриптор System V IPC действителен в рамках всей операционной системы, и передать его значение от процесса, создавшего разделяемую память, текущему процессу. Отметим, что при создании разделяемой памяти с помощью значения *IPC_PRIVATE* — это единственно возможный способ.

После получения дескриптора необходимо включить область разделяемой памяти в адресное пространство текущего процесса. Это осуществляется с помощью системного вызова *shmat()*. При нормальном завершении он вернет адрес разделяемой памяти в адресном пространстве текущего процесса. Дальнейший доступ к этой памяти осуществляется с помощью обычных средств языка программирования.

Системный вызов *shmat()*

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

Описание системного вызова

Системный вызов *shmat* предназначен для включения области разделяемой памяти в адресное пространство текущего процесса. Данное описание не является полным описанием системного вызова, а ограничивается рамками текущего курса. Для полного описания обращайтесь UNIX Manual.

Параметр *shmid* является дескриптором System V IPC для сегмента разделяемой памяти, т. е. значением, которое вернул системный вызов *shmget()* при создании сегмента или при его поиске по ключу.

В качестве параметра *shmaddr* в рамках нашего курса мы всегда будем передавать значение NULL, позволяя операционной системе самой разместить разделяемую память в адресном пространстве нашего процесса.

Параметр *shmflg* в нашем курсе может принимать только два значения: 0 - для осуществления операций чтения и записи над сегментом и *SHM_RDONLY* - если мы хотим только читать из него. При этом процесс должен иметь соответствующие права доступа к сегменту.

Возвращаемое значение

Системный вызов возвращает адрес сегмента разделяемой памяти в адресном пространстве процесса при нормальном завершении и значение -1 при возникновении ошибки.

После окончания использования разделяемой памяти процесс может уменьшить размер своего адресного пространства, исключив из него эту область с помощью системного вызова *shmdt()*. Отметим, что в качестве параметра системный вызов *shmdt()* требует адрес начала области разделяемой памяти в адресном пространстве процесса, т. е. значение, которое вернул системный вызов

*shmat()*, поэтому данное значение следует сохранять на протяжении всего времени использования разделяемой памяти.

Прототип системного вызова *shmdt()*

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int shmdt (char *shnaddr),"

Описание системного вызова

Системный вызов *shmdt* предназначен для исключения области разделяемой памяти из адресного пространства текущего процесса.

Параметр *shmaddr* является адресом сегмента разделяемой памяти, т. е. значением, которое вернул системный вызов *shmat()*.

Возвращаемое значение

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

Для иллюстрации использования разделяемой памяти давайте рассмотрим две взаимодействующие программы:

*/* Программа 1 */*

*/* Мы организуем разделяемую память для массива из трех целых чисел. Первый элемент массива является счетчиком числа запусков программы 1, т. е. данной программы, второй элемент массива - счетчиком числа запусков программы 2, третий элемент массива - счетчиком числа запусков обеих программ */*

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <sys/shm.h>*

*#include <stdio.h>*

*#include <errno.h>*

```
int main() {
int  *array;   /* Указатель на разделяемую память */
int  shmid;   /*  IPC дескриптор для области разделяемой памяти */
int new =  1;   /* Флаг необходимости инициализации элементов массива */
char pathname[]   =  "06-la.c";   /* Имя файла, используемое для генерации ключа.  Файл с таким именем должен существовать в текущей директории */
key_t key;   /*  IPC ключ */
/*Генерируем IPC ключ из имени файла 06-1a.c в
текущей директории и номера экземпляра области
разделяемой памяти 0  */
if((key =  ftok(pathname, 0) )   <  0){
printf("Can\'t generate key\n");
exit(-1);
}
/* Пытаемся эксклюзивно создать разделяемую память для сгенерированного ключа,  т.е.  если для этого ключа она уже существует, системный вызов вернет отрицательное значение.  Размер памяти определяем как размер массива из трех целых переменных,  права доступа 0666 - чтение и запись разрешены для всех */
if((shmid =  shmget(key,   3 * sizeof(int) , 0666|IPC_CREAT|IPC_EXCL))   <  0){
/* В случае ошибки пытаемся определить:  возникла ли она из-за того,  что сегмент разделяемой памяти уже существует или по другой причине */
if(errno   ! = EEXIST) {
/* Если по другой причине - прекращаем работу */
printf("Can\'t create shared memory\n");
exit(-1); }  else  {
/*  Если из-за того,  что разделяемая память уже существует,   то пытаемся получить ее IPC дескриптор и,  в случае удачи,  сбрасываем флаг необходимости инициализации элементов массива */
```

```
if((shmid = shmget(key,  3*sizeof(int),  0))  < 0){
printf("Can\'t  find shared memory\n"); exit(-1);
}
new =  0;
}
/* Пытаемся отобразить разделяемую память в адресное пространство текущего процесса.  Обратите внимание на то,  что для правильного сравнения мы явно преобразовываем значение -1 к указателю на целое.*/
if((array = (int *)shmat(shmid, NULL, 0)) == (int *)(-l)){
printf("Can't attach shared memory\n");
exit(-1);
}
/* В зависимости от значения флага new либо инициализируем массив, либо увеличиваем соответствующие счетчики */
if(new){
array[0]   =  1 ;
array[1]   =  0;
array[2]   =  1;
}  else  {
array[0]   +=  1 ;
array[2]   += 1;
}
/* Печатаем новые значения счетчиков, удаляем разделяемую память из адресного пространства текущего процесса и завершаем работу */
printf("Program 1 was spawn %d times,
program 2 - %d times,  total - %d times\n",
array[0],  array[1],  array[2]); if(shmdt(array)  < 0){
printf("Can't detach shared memory\n");
exit(-1);
}
```

```
return 0;
}
/* Программа 2 */
/* Мы организуем разделяемую память для массива из трех целых чисел.
Первый элемент массива является счетчиком числа запусков программы 1, т. е.
данной программы, второй элемент массива - счетчиком числа запусков
программы 2, третий элемент массива -счетчиком числа запусков обеих
программ */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
int main() {
int *array; /* Указатель на разделяемую память */
int shmid; /* IPC дескриптор для области разделяемой памяти */
int new =1; /* Флаг необходимости инициализации элементов массива */
char pathname[] = "06-la.c"; /* Имя файла, используемое для генерации
ключа. Файл с таким именем должен существовать в текущей директории */
key_t key; /* IPC ключ */
/* Генерируем IPC ключ из имени файла 06-1a.c в текущей директории и
номера экземпляра области разделяемой памяти 0 */
if((key = ftok(pathname,0)) < 0){
printf("Can\'t generate key\n");
exit(-1);
}
/* Пытаемся эксклюзивно создать разделяемую память для
сгенерированного ключа, т. е. если для этого ключа она уже существует,
системный вызов вернет отрицательное значение. Размер памяти определяем
```

*как размер массива из трех целых переменных, права доступа 0666 - чтение и запись разрешены для всех */*

*if((shmid = shmget(key, 3 *sizeof(int), 0666 IIPC_CREATIIPC_EXCL)) < 0){*

*/* В случае возникновения ошибки пытаемся определить: возникла ли она из-за того, что сегмент разделяемой памяти уже существует или по другой причине */*

*if(errno != EEXIST){*

*/* Если по другой причине - прекращаем работу */*

*printf("Can\'t create shared memory\n"); exit(-1) ;*

*} else {*

*/* Если из-за того, что разделяемая память уже существует, то пытаемся получить ее IPC дескриптор и, в случае удачи, сбрасываем флаг необходимости инициализации элементов массива */*

*if((shmid = shmget(key, 3*sizeof(int), 0)) < 0){*

*printf("Can\'t find shared memory\n"); exit(-1);*

*}*

*new = 0 ;*

*}*

*}*

*/* Пытаемся отобразить разделяемую память в адресное пространство текущего процесса. Обратите внимание на то, что для правильного сравнения мы явно преобразовываем значение -1 к указателю на целое.*/*

*if ((array = (int *)shmat(shmid, NULL, 0)) = = (int *) (-1) ) {*

*printf("Can't attach shared memory\n"); exit(-1);*

*}*

*/* В зависимости от значения флага new либо инициализируем массив, либо увеличиваем соответствующие счетчики */*

*if(new){*

*array[0] = 0 ;*

*array[1] = 1;*

```
array[2]  = 1; }  else  {
array[1]  += 1;
array[2]  += 1;
}
/* Печатаем новые значения счетчиков,  удаляем разделяемую память из адресного пространства текущего процесса и завершаем работу */
printf("Program 1 was spawn %d times,
program 2  -  %d times,   total  -  %d times\n", array[0],  array[1],  array[2]);
if(shmdt(array)  <  0){
printf("Can't detach shared memory\n");
exit(-1) ;
}
return 0 ;
}
```

Эти программы очень похожи друг на друга и используют разделяемую память для хранения числа запусков каждой из программ и их суммы.

В разделяемой памяти размещается массив из трех целых чисел. Первый элемент массива используется как счетчик для программы 1, второй элемент — для программы 2, третий элемент — для обеих программ суммарно. Дополнительный нюанс в программах возникает из-за необходимости инициализации элементов массива при создании разделяемой памяти. Для этого нам нужно, чтобы программы могли различать случай, когда они создали ее, и случай, когда она уже существовала. Мы добиваемся различия, используя вначале системный вызов *shmget()* с флагами *IPC_CREAT* и *IPC_EXCL*. Если вызов завершается нормально, то мы создали разделяемую память. Если вызов завершается с констатацией ошибки и значение переменной *errno* равняется *EEXIST*, то, значит, разделяемая память уже существует, и мы можем получить ее IPC-дескриптор, применяя тот же самый вызов с нулевым значением флагов.

**Команды ipcs и ipcrm**

Как мы видели из предыдущего примера, созданная область разделяемой памяти сохраняется в операционной системе даже тогда, когда нет ни одного процесса, включающего ее в свое адресное пространство. С одной стороны, это имеет определенные преимущества, поскольку не требует одновременного существования взаимодействующих процессов, с другой стороны, может причинять существенные неудобства. Допустим, что предыдущие программы мы хотим использовать таким образом, чтобы подсчитывать количество запусков в течение одного, текущего, сеанса работы в системе. Однако в созданном сегменте разделяемой памяти остается информация от предыдущего сеанса, и программы будут выдавать общее количество запусков за все время работы с момента загрузки операционной системы. Можно было бы создавать для нового сеанса новый сегмент разделяемой памяти, но количество ресурсов в системе не безгранично. Нас спасает то, что существуют способы удалять неиспользуемые ресурсы System V IPC как с помощью команд операционной системы, так и с помощью системных вызовов. Все средства System VI PC требуют определенных действий для освобождения занимаемых ресурсов после окончания взаимодействия процессов. Для того чтобы удалять ресурсы System V IPC из командной строки, нам понадобятся две команды, *ipcs* и *ipcrm*.

Команда *ipcs* выдает информацию обо всех средствах System V IPC, существующих в системе, для которых пользователь обладает правами на чтение: областях разделяемой памяти, семафорах и очередях сообщений.

Синтаксис команды *ipcs*

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

Описание команды

Команде *ipcs* предназначена для получения информации о средствах System V IPC, к которым пользователь имеет право доступа на чтение.

Опция *-i* позволяет указать идентификатор ресурсов. Будет выдаваться только информация для ресурсов, имеющих этот идентификатор.

Виды IPC ресурсов могут быть заданы с помощью следующих опций:

*-s* для семафоров;

*-m* для сегментов разделяемой памяти;

*-q* для очередей сообщений;

*-a* для всех ресурсов (по умолчанию).

Опции *[-tclup]* используются для изменения состава выходной информации. По умолчанию для каждого средства выводятся его ключ, идентификатор IPC, идентификатор владельца, права доступа и ряд других характеристик. Применение опций позволяет вывести;

*-t* времена совершения последних операций над средствами IPC;

*-p* идентификаторы процесса, создавшего ресурс, и процесса, совершившего над ним последнюю операцию;

*-c* идентификаторы пользователя и группы для создателя ресурса и его собственника

*-l* системные ограничения для средств System V IPC;

*-u* общее состояние IPC ресурсов в системе.

Опция *-h* используется для получения краткой справочной информации.

Из всего многообразия выводимой информации нас будут интересовать только IPC идентификаторы для средств, созданных вами. Эти идентификаторы будут использоваться в команде *ipcrm*, позволяющей удалить необходимый ресурс из системы. Для удаления сегмента разделяемой памяти эта команда имеет вид

*ipcrm shm <1PC идентификатор>*

Удалите созданный вами сегмент разделяемой памяти из операционной системы, используя эти команды.

Синтаксис команды ipcrm

*ipcrm [shm | msg | sem] id*

Описание команды

Команда *ipcrm* предназначена для удаления ресурса System V IPC из операционной системы. Параметр *id* задает IPC-идентификатор для удаляемого ресурса, параметр *shm* используется для сегментов разделяемой памяти, параметр *msg* - для очередей сообщений, параметр *sem* - для семафоров.

Если поведение программ, использующих средства System V IPC, базируется на предположении, что эти средства были созданы при их работе, не забывайте перед их запуском удалять уже существующие ресурсы.

**Использование системного вызова shmctl() для освобождения ресурса**

Для той же цели — удалить область разделяемой памяти из системы — можно воспользоваться и системным вызовом *shmctl()*. Этот системный вызов позволяет полностью ликвидировать область разделяемой памяти в операционной системе по заданному дескриптору средства IPC, если, конечно, у вас хватает для этого полномочий. Системный вызов *shmctl()* позволяет выполнять и другие действия над сегментом разделяемой памяти, но их изучение лежит за пределами нашего курса.

Прототип системного вызова *shmctl()*

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <sys/shm.h>*

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

Описание системного вызова

Системный вызов *shmctl* предназначен для получения информации об области разделяемой памяти, изменения ее атрибутов и удаления из системы. Данное описание не является полным описанием системного вызова, а

ограничивается рамками текущего курса. Для изучения полного описания обращайтесь к UNIX Manual.

В нашем курсе мы будем пользоваться системным вызовом *shmctl* только для удаления области разделяемой памяти из системы. Параметр *shmid* является дескриптором System V IPC для сегмента разделяемой памяти, т. е. значением, которое вернул системный вызов *shmget()* при создании сегмента или при его поиске по ключу.

В качестве параметра *and* в рамках нашего курса мы всегда будем передавать значение *IPC_RMID* - команду для удаления сегмента разделяемой памяти с заданным идентификатором. Параметр *buf* для этой команды не используется, поэтому мы всегда будем подставлять туда значение *null*.

Возвращаемое значение

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

**Разделяемая память и системные вызовы fork(), exec() и функция exit()**

Важным вопросом является поведение сегментов разделяемой памяти при выполнении процессом системных вызовов *fork(), exec()* и функции *exit()*.

При выполнении системного вызова *fork()* все области разделяемой памяти, размещенные в адресном пространстве процесса, наследуются порожденным процессом.

При выполнении системных вызовов *exec()* и функции *exit()* все области разделяемой памяти, размещенные в адресном пространстве процесса, исключаются из его адресного пространства, но продолжают существовать в операционной системе.

## 2. Практические задания

### Задание 1. Взаимодействие через разделяемую память

Напишите две программы, осуществляющие взаимодействие через разделяемую память. Первая программа должна создавать сегмент разделяемой памяти и копировать туда собственный исходный текст, вторая программа должна брать оттуда этот текст, печатать его на экране и удалять сегмент разделяемой памяти из системы.

**Литература**

1. В.Е. Карпов, К.А. Коньков Основы операционных систем. Курс лекций. Учебное пособие. Издание второе, дополненное и исправленное. М.: ООО «ИНТУИТ.ру» 2005