Semaphores_definition

Семафоры

являются средством синхронизации доступа процессов к разделяемым ресурсам.

InterProcessCommunications

В системе **IPC** Linux особое значение имеют три технологии:

- очереди сообщений (Message Queue),
- **семафоры** (Semaphores),
- разделяемая память (Shared Memory).

Общее:

- Объекты IPC используются совместно произвольными процессами.
- Остаются существовать в системе даже после завершения этих процессов.
- Процедура назначения имен объектам IPC нетривиальна.
- Каждый объект имеет свой уникальный идентификатор (дескриптор).
- Уникальность дескриптора обеспечивается внутри типа объектов IPC.
- Работа со всеми 3-мя видами средств в определенной степени унифицирована.

IPC_ftok() unification

Имя для объекта IPC называется ключом **key**
и генерируется функцией **ftok**():

```
#include <sys/types.h>
#include <sys/ipc.h>
...
key_t ftok (char *filename, char proj);      // filename - имя некоторого файла,
                                            // известного всем взаимодействующим
        // процессам (стабильного).            proj - идентификатор проекта.
```

Унификация работы с IPC:

- ✔ вызов **semget**() для создания объекта (получения доступа),
- ✔ флаги создания объекта **semflag**
- ✔ вызов **semctl**() для управления объектами.

IPC_get() ipcflag ctl()

msget(), **msgctl**() - message queue

semget(), **semctl**() - **semaphores**

shmget(), **shmctl**() - shared memory

Переменная **semflag** определяет **права доступа** к объекту и указывает

- создается ли новый объект ***IPC_CREATE*** или
- требуется доступ к существующему ***IPC_EXCL***.
- Создается заново при каждом запуске ***IPC_PRIVATE*** .

Операции над созданными объектами IPC (помещение/получение сообщения, установка семафоров, чтение/запись в разделяемую память) производятся с помощью других системных вызовов, также **унифицированных**.

IPC_manage

- Для каждого из созданных IPC объектов ядро операционной системы поддерживает внутреннюю *системную структуру* данных.
- Управляются поля структуры вызовами типа **ctl**().
- Операционная система не удаляет созданные объекты IPC даже, когда ни один процесс не пользуется ими.
- Удаление созданных объектов IPC дело самих процессов. Которые должны “договориться” об этом.

Semaphores_functionality

Функция семафоров – разрешать или запрещать процессам использование того или иного разделяемого ресурса.

Например, значение счетчика 0 – ресурс занят и операция недопустима, процессу остается ждать.
Значение счетчика 1 – ресурс доступен, можно работать с ресурсом, заблокировав его, установкой счетчика в 0.

После выполнения процессом операции необходимо освободить ресурс, а для этого значение счетчика необходимо изменить на 1.

В этом примере счетчик играет роль семафора.

Semaphores_functionality

Необходимо обеспечить выполнение следующих условий:

- Значение семафора должно быть доступно различным процессам (значит, должен находиться в адресном пространстве ядра).
- Операция проверки и изменения значения семафора должна быть реализована в виде одной **атомарной** по отношению к другим процессам операции (непрерываемой другими процессами).
- Единственным способом гарантировать атомарность **критических участков** операций является выполнение этих операций в **режиме ядра**.
- Поэтому семафоры являются **системным ресурсом**, действия над которым производятся через интерфейс системных вызовов.

Semaphores_Linux features

Семафоры в составе средств IPC Linux, обладают характеристиками:

- Семафор представляет собой не один счетчик, а **группу**, состоящую из нескольких счетчиков, объединенных общими признаками (дескриптором объекта, правами доступа и т.д.)
- Каждый из этих счетчиков может принимать **любое неотрицательное** значение в пределах, определенных системой (а не только значения 0 и 1).
- Для каждой группы семафоров ядро поддерживает структуру данных **semid_ds**.
- Значение конкретного семафора из набора хранится во внутренней структуре **sem**.

Semaphores_structure semid_ds

Поля системной структуры **semid_ds**

struct ipc_perm sem_perm	Описание прав доступа
*struct sem *sem_base*	Указатель на первый элемент массива семафоров
ushort sem_nsems	Число семафоров в группе
time_t sem_otime	Время последней операции
time_t sem_ctime	Время последнего изменения

Semaphores_structure sem

Поля системной структуры **sem**

ushort semval	Значение семафора
pid_t sempid	Идентификатор процесса, выполнившего последнюю операцию над семафором
ushort semncnt	Число процессов, ожидающих увеличения значения семафора
ushort semzcnt	Число процессов, ожидающих обнуления семафора

Помимо собственно **значения семафора**, в структуре **sem** хранится информация, позволяющая ядру производить операции над семафорами.

Semaphores_semget()

Для создания и для получения доступа к семафору используется **semget**()

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

int semget (key_t key, int nsems, int semflags);
```

nsems – число семафоров в группе.
В случае получения доступа к уже существующему семафору **nsem** игнорируется.

Semaphores_generation

В примере создается 3 набора
семафоров с разными флагами:

```
/* Программа gener_sem.cpp */

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

Semaphores_generation

```
main(void)
{
int
sem1, sem2, sem3;
key_t ipc_key;
ipc_key = ftok(“.”, `S`);
if ((sem1=semget(ipc_key, 3, IPC_CREAT | 0666)) == -1){
perror("semget: IPC_CREAT | 0666");
}
printf("sem1 identifier is %d\n", sem1);
if((sem2 = semget(ipc_key, 3, IPC_CREATE | IPC_EXCL | 0666))
== -1) {
perror("semget: IPC_CREATE | IPC_EXCL | 0666");
}
printf("sem2 identifier is %d\n", sem2);
if((sem3 = semget(IPC_PRIVATE, 3, 0600)) == -1) {
perror("semget: IPC_PRIVATE");
}
printf("sem3 identifier is %d\n", sem3);
exit(0);
}
```

Semaphores_generation

При запуске этой программы многократно, видно, что:

- набор **sem1** будет создан единожды, а затем каждая новая попытка будет всего лишь открывать доступ к уже существующему ресурсу;
- попытки создания набора **sem2** на том же ключе всегда будут приводить к ошибке из-за наличия флагов IPC_CREATE | IPC_EXCL, не допускающих открытия ресурса вместо его создания;
- набор **sem3** будет создаваться при каждом новом запуске программы (флаг IPC_ PRIVATE). Причем каждый раз с новым уникальным идентификатором.

Просмотреть список созданных семафоров и прав доступа можно командой **ipcs**.

Semaphores_operations

После получения дескриптора семафора процесс может производить над семафором операции системным вызовом

```
#include <sys/itypes.h>
#include <sys/ipc.h>
#include<sys/sem.h>
int semop (int semid,  struct sembuf *semop,  size_t nops);
```

- **semid -** идентификатор (дескриптор) семафора, возвращенный предыдущим вызовом **semget**();
- второй параметр – указатель на структуру, определяющую требуемые операции;
- **nops** - число операций над семафором

Semaphores_struct sembuf

Поля системной структуры **sembuf**

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

Поле **sem_op** - определяет 3 возможные операции над семафором.

Поле **sem_flg** - флаг, обычно установленный в SEM_UNDO, если процесс завершается без освобождения семафора (операция остается блокированной), то это позволяет семафору сброситься автоматически (отменить операцию).

Ядро обеспечивает атомарность выполнения критических участков операций.

Semaphores_struct sembuf.sem_op

Кодирование операций в поле **sem_op**:

1. если значение **sem_op** равно нулю, процесс ожидает, пока семафор не обнулится (т.е. происходит проверка значения семафора);
2. если величина **sem_op** положительна, то текущее значение семафора увеличивается на эту величину;
3. если величина **sem_op** отрицательна, процесс ожидает, пока значение семафора не станет большим или равным абсолютной величине sem_op, а затем абсолютная величина semop вычитается из значения семафора).

При работе с семафорами процессы используют различные комбинации из 3-х операций, по своему трактуя значения семафоров (процессы вольны решать, какое именно значение семафора является *разрешающим).*

Shared Memory

Разделяемая память

Предоставляет ***наиболее быстрый***
способ обмена данными между независимыми процессами,

но не самый простой по организации.

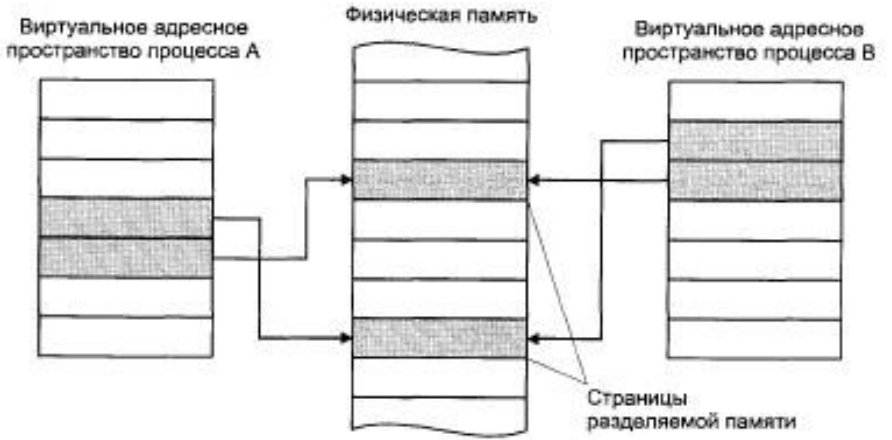
Shared Memory Scenario

Примерный сценарий работы с разделяемой памятью:

1. Сервер *создает сегмент* разделяемой (общей) памяти, задавая его *размер* и *права доступа* к нему.
2. *Присоединяет* этот сегмент к своей памяти, отображая его при этом на область данных.
3. Сервер получает доступ к разделяемой памяти, используя семафор, производит *запись* данных в разделяемую память.
4. После завершения записи сервер *освобождает* разделяемую память с помощью семафора.
5. Другой процесс (клиент), получающий права доступа к сегменту, может также *отобразить* его на свою область данных.
   Клиент при получении *доступа* к разделяемой памяти, *запирает* ресурс с помощью семафора.
6. Клиент производит *чтение* данных из разделяемой памяти и *освобождает* ее, используя семафор.

Shared Memory

Несколько процессов отображают область разделяемой памяти в различные участки собственного виртуального адресного пространства:



По окончании коммуникаций через сегмент общей памяти, обычно *процесс владелец* отвечает за *удаление* этого сегмента.

Процесс владелец (создавший сегмент общей памяти) может *делегировать* свои *полномочия* и другому процессу.

Shared Memory Structure

Создание сегмента Shared Memory

Для каждой создаваемой области разделяемой памяти, ядро поддерживает структуру данных **shmid_ds** (определена в <sys/shm.h>),

основными полями которой являются:

struct **ipc_perm**	**shm_perm**	- Права доступа, владельца области.
Int	**shm_segsz**	- Размер выделяемой памяти.
ushort	**shm_nattch**	- Число процессов, использующих разделяемую память.
time_t	**shm_atime**	- Время последнего присоединения к разделяемой памяти.
time_t	**shm_dtime**	- Время последнего отключения от разделяемой памяти.
time_t	**shm_ctime**	- Время последнего изменения.

Shared Memory Creation

Создание сегмента Shared Memory

shmget()

возвращает уникальный идентификатор (дескриптор)
разделяемого сегмента памяти
(в случае неудачи -1).

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

int shmget (key_t, int size, int shmflag);
```

Флаги *IPC_PRIVATE* *IPC_CREAT* *IPC_EXCL*

Shared Memory Creation Sample

Пример создания **двух сегментов** разделяемой памяти *gener_shma.cpp*:

```
main(void)
{
key_t key=15;

Int  shmid_1, shmid_2;

if ((shmid_1=shmget(key, 1000, 0644|IPC_CREAT)) == -1){
       perror("shmget shmid_1");
       exit(1);
       }

printf("First memory identifir e is %d \n", shmid_1);
     if((shmid_2 = shmget(IPC_PRIVATE, 20, 0644)) == -1) {
        perror("shmget shmid_2");
        exit(2);
        }
printf("Second shared memory identifire is %d \n", shmid_2);
exit(0);
}
```

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

Shared Memory Error Codes

В случае неуспешного выполнения, ***shmget***() возвращает *-1* и устанавливает переменную **errno** равную коду ошибки, например:

EONT - идентификатор разделяемой памяти не существует для данного *key* и *IPC_CREAT* не установлен;

EACC - идентификатор разделяемой памяти существует для данного *key*, но данная операция запрещена текущими правами доступа;

EEXIST - идентификатор разделяемой памяти существует для данного *key*, но установлены флаги *IPC_CREAT* и *IPC_EXCL*;

ENOMEM - недостаточно памяти для выполнения операции;

EINVAL - неверный размер сегмента;

ENOSPC - достигнут предел числа разделяемых сегментов системы.

Shared Memory Limits

Ограничения для сегмента разделяемой памяти :

Максимальный размер сегмента	1,048,576 байт
Минимальный размер сегмента	1 байт
Максимальное число сегментов в системе	100
Максимальное число сегментов, связанных с процессом	6

Shared Memory Attach/Detach

Присоединение и отсоединение

Для работы с разделяемой памятью (чтение и запись) необходимо сначала *присоединить* (attach) область вызовом **shmat**():

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

*(char *)* **shmat** *(int shmid, char *shmaddr, int shmflag);*

Вызов **shmat**() возвращает *адрес* начала области в пространстве процесса размером *size ,* заданном в предшествующем вызовом **shmget**() .

Shared Memory Attach

Правила получения этого *адреса* :

1. Если аргумент *shmaddr* **нулевой**, то система **самостоятельно** выбирает адрес.
2. Если аргумент *shmaddr* отличен от нуля, значение возвращаемого адреса зависит от наличия флажка *SHM_RND* в аргументе *shmflag*:
   - Если флажок *SHM_RND* не установлен, система присоединяет разделяемую память к указанному *shmaddr* адресу.
   - Если флажок *SHM_RND* установлен, система присоединяет разделяемую память к адресу, полученному округлением в меньшую сторону *shmaddr* до некоторой определенной величины *SHMLBA* (используется системой как размер страницы памяти).

По умолчанию разделяемая память присоединяется с правами на ***чтение*** и ***запись***. Это можно изменить флажком *SHM_RDONLY* .

Shared Memory Detach

Окончив работу с разделяемой памятью, процесс *отключает* (detach) область разделяемой памяти вызовом **shmdt**():

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int **shmdt** *(char *shmaddr);*

Программа *attach_shm.cpp* иллюстрирует процедуры *присоединения* сегмента общей памяти, *отображения* ее на память процесса и *операции* с ней.

Shared Memory Attach/Operate/Detach

```
main(void) {
    pid_t  pid;
    int shmid;
    char   c, *shm, *s;

    if ((shmid=shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT|0666))< 0) {
            perror("shmget fail");
            exit(1);
    }
    if ((shm = (char *) shmat( shmid, 0, 0)) == (char *) -1) {
            perror("shmat : parent ");
            exit(2);
    }
    printf("Addresses in parent\n\n");
    printf("shared mem: %X etext: %X edata: %X end: %X\n\n",
            shm, &etext, &edata, &end);

s = shm;                        /* Указатель s ссылается на разделяемую память */
for (c = 'A'; c <= 'Z'; ++c)     /* Выложить в область разделяемой памяти */
      *s++ = c;
*s = NULL;                    /* Маркер конца строки */
printf("In parent before fork, memory is : %s \n", shm);
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHM_SIZE 30
extern etext, edata, end;
```

Shared Memory Attach/Operate/Detach

```
    pid = fork();

switch (pid) {
    case -1:
        perror("fork ");
        exit(3);
    default:
        sleep(5);              /* Ожидание завершения дочернего процесса */
        printf("\nIn parent after fork, memory is  : %s\n", shm);
        printf("Parent removing shared memory\n");
        shmdt(shm);
        shmctl(shmid, IPC_RMID, (struct shmid_ds *) 0 );
        exit(0);
    case 0:
        printf("In child after fork, memory is : %s \n",shm);
        for (; *shm; ++shm) /* Модификация разделяемой памяти */
        *shm += 32;
        shmdt (shm );
        exit(0);
    }
 }
```

Shared Memory Attach/Operate/Detach

В программе *создаётся* новый сегмент разделяемой памяти длиной 30 байт.
Этот сегмент *привязывается* к области данных процесса, используя первый доступный адрес.
Реальные адреса привязки, хранимые в *etext, edata, end*, выводятся на консоль.
Указатель типа *char** содержит адрес начала сегмента разделяемой памяти, в который *записываются данные* (последовательность прописных букв).
Системный вызов *fork* создаёт процесс-потомок, который повторно выводит на экран содержимое разделяемого сегмента, а затем *модифицирует* содержимое разделяемой памяти (преобразованием прописных букв в строчные).
По окончании преобразования, процесс-потомок *отсоединяет* разделяемый сегмент и завершает работу.
Процесс-родитель, после 5-ти секундного ожидания, выводит на консоль содержимое разделяемого сегмента, *отсоединяет* сегмент разделяемой памяти и *удаляет* его системным вызовом *shmctl()*.

Thanks for your attention

Спасибо за внимание !

vladimir.shmakov.2012@gmail.com