# Семинар 5 (29.11.2021)

void**\*** shmat**(**int shmid**,** const mid **\***shmaddr**,** int shmflg**);**

const mid **\***shmaddr = 0 SHM\_RND

На лекции рассматривали проблемы параллельных процессов.

## Средства взаимодействия процессов

## IPC System V

IPC – inter process communication – меж процессорное взаимодействие.

В System V используются следующие средства меж процессорного взаимодействия:

1. Сигналы;
2. Семафоры;
3. Программные каналы (именованные и не именованные)
4. Сегменты разделяемой памяти
5. Очереди сообщений. (В unix bsd это сокеты – след. семестр)

Unix – это linux подобная ОС.

Выделяются файлы отображаемые в память (mapping file) – но мы их рассматривать не будем.

### Для 5ой ЛР: Симафоры UNIX и Сегменты разделяемой памяти

***N1*** Семафоры не имеют хозяина => mutix может освободить…

Семафор может освободить любой процесс, который знает его идентификатор.

***N2*** команды … и на эти команды непроизводительно тратится процессорное время. Введение семафоров … в режим ядра. … Это же привело к крайне негативной ситуации – если процессы используют большое кол-во семафоров, то очень сложно уследить за их выполнением. Поэтому всегда было стремление структурировать это… Использование набора семафоров является некоторым решением этой проблемы. Unix поддерживают наборы семафоров.

В ядре системе имеется таблица семафоров. В этой таблице отслеживаются все созданные в системе семафоры. (struct semid\_ds <sys/sem.h>) Фактически это таблица дескрипторов семафоров, каждая строка описывает отдельный набор. О каждом наборе известно:

1. Имя (целое число, в Unix все идентефикаторы целые числа) – присваивается процессом который создал набор. Другие процессы, по этому имени, могут ?открыть…? и получить дескриптор для доступа к набору.  
   **Одной неделимой операцией можно изменить все или часть набора семафоров.**
2. U\_ID и G\_ID (Создателя и группы)

Процесс U\_ID которого совпадает с U\_ID ?процесс может изменять его и …?

1. Права доступа (Users, Group, Others, Read/Write/Execute)
2. Кол-во семафоров в наборе
3. Время изменения одного или нескольких значений семафоров последним процессом

Это важно в отношении: случилось до, случилось после.

1. Время последнего изменения управляющих … набора, управляющим процессом. (По той же причине)
2. Указатель на набор (или массив) семафоров.

Индексы начинаются с 0.

О каждом семафоре набора имеются след. данные:

1. Значение семафора
2. Идентификатор процесса, который оперировал семафором в последний раз.
3. Число процессов, заблокированных в текущий момент времени на семафоре.

Для семафоров определены следующие системные вызовы:  
semget();  
?semctl();  
semop();

void**\*** shmat**(**int shmid**,** const mid **\***shmaddr**,** int shmflg**);**

int semget**(**key\_t key**,** int num\_sem**,** int flg**);** //верн1т дескриптор

//sem control - управлять

int semgctl**(**int semfd**,** int num**,** int cmd**,** union semun arg**);** //fd - file descriptor

int semop**(**int semfd**,** struct semluf **\***opsptr**,** int lim**);**

Если системный вызов не выполнен, то он возвращает -1 => их проверяют на -1.

struct semluf

**{**

nshort sem\_num**;** //

short sem\_op**;** //Операция на семафоре

short sem\_fl**;** //Флаги определённые на семафоре

**}**

Три типа операций:

1. sem\_op > 0 – освобождение семафора
2. sem\_op == 0 – процесс выполнивший такое, переводится в сост. Ожидания до момента освобождения ресурса. (Блокируется)
3. sem\_op < 0 – захват семафора (⬄ P на S)

Флаги:

* IPC\_NOWAIT – флаг информирует ядро о нежелании процесса переходить в состояние ожидания. Наличие этого флага объясняется желание избежать блокировки всех процессов, которые находятся в очереди к семафору. В случае если, захвативший семафор процесс завершился аварийно или получил сигнал kill. В силу того что сигнал kill нельзя перехватить, процесс не сможет освободить семафор. И все процессы которые находятся в очереди к данному семафору будут заблокированы на вечно (что очень негативно).
* SEM\_UNDO – этот флаг указывает ядру, что необходимо отслеживать изменения значений семафора в результате вызова semop() для того чтобы при завершении процесса ядро могло ликвидировать сделанные процессом изменения. (Для семафора это важно – у него нет хозяина)

Пример:

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

struct sem\_but sbuf**[**2**]={{**0**,** **-**1**,** SEM\_UNDO**|**IPC\_NOWAIT**},{**1**,** 0**,** 1**}};** //Массив структур

int main**()**

**{**

int perms **=** S\_IRWXV**|**S\_IRWXG**|**S\_IRWX\_0**;** //Полные права доступа

int fd segment**(**100**,** 2**,** IPC\_CREATE**|**perms**);** //Создаёт набор из двух семафоров с идентификатором 100, возвращает файловый дескриптор

**if** **(**fd **==** **-**1**)** //Набор создать не удалось

**{**

perror**(**"semop"**);**

exit**(**1**);**

**}**

//Если удачно, то выполняем semop()

**if** **(**semop**(**fd**,** sbuf**,** 2**)** **==** **-**1**)** //передаём

perror**(**"semop"**);**

//Если всё благополучно то

//То есть семафор был захвачен, не освобождён, процесс завершился

//Но система отменит изменения т.к. установлены флаги

**return** 0**;**

**}**

### Сегменты разделяемой памяти

Разделяемая память – один процесс может туда записать значение, а другой считать значение оттуда.

В ядре имеется таблица сегментов разделяемой памяти. (Таблица разделяемых сегментов) (struct shmid\_ds <sys/shm.h>)

Ни один процесс не может залезть в чужое адресное пространство, поэтому для общения нужно третье. Разделяемые сегменты подключаются к виртуальному адресному пространству, то есть получает ссылку на разделяемый сегмент.

На разделемых сегментах определены следующие системные вызовы:  
Shmget();  
Shmctl();  
Shmat(); // touch? - подключить  
Shmdt(); //detouch – отключить

После создания разделяемого сегмента, любой процесс может присоединить его к своему виртуальному адресному пространству (исп. с.в. Shmat()) и работать с ним к с собственным. По завершении процесса, разделяемый сегмент сохраняется. То есть разделяемые сегменты не удаляются даже если завершаются процессы, которые их создали. У каждой разделяемой памяти (или разделяемого сегмента) есть назначенный владелец и удалять эту область из ядра или корректировать её управляющие параметры могут процессы, которые имеют привилегированные права, создателя или назначенного владельца.

Пример:

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

int main**()**

**{**

int peerms **=** S\_IRWXU**|**S\_IRWXG**|**S\_IRWXO**;**

//Создаётся роазделяемый сегменты с идентефикатором 100, 1024 байт

//С полными правами

int fd **=** shmget**(**100**,** 1024**,** IPC\_CREATE**|**perms**);**

**if** **(**fd **==** **-**1**)**

**{**

perror**(**"shmset"**);**

exit**(**1**);**

**}**

//Если создан, процесс пытается подключить разд. сегмент

cahr **\***addr **=** **(**char**\*)**shmat**(**fd**,** 0**,** 0**);**

**if** **(**addr **==** **(**char**\*)-**1**)**

**{**

perror**(**"shmat"**);**

exit**(**1**);**

**}**

//Если получилось подключить, он записывает туда "Hello"

strcpy**(**addr**,** "Hello"**);**

//Затем процесс отсоединяется от разделяемого сегмента

**if** **(**shmdt**(**addr**)==-**1**)**

perror**(**"shmdt"**);**

**return** 0**;**

**}**

void **\***shmat**(**int shmid**,** const void **\***shmaddr**,** int shmflg**);**

В системе на (?Разделённых сегментах?)… определены следующие системные ограничения:

|  |  |
| --- | --- |
| Системные ограничения | Знач. |
| SHMMNI | Максимально возможное кол-во сегментов, которые могут существовать одновременно |
| SHMMIN | минимально возможный размер сегмента в байтах |
| SHMMAX | максимально возможный размер сегмента в байтах |

*SHMMNI – кол-во сегментов … могут существовать одновременно*

*SHMMIN – минимально возможный размер сегмента в байтах*

*SHMMAX – максимально возможный размер сегмента в байтах*

Причём, если процесс пытается создать сегмент, когда их уже максимально возможное кол-во, то процесс блокируется.

Процесс пытающийся создать сегмент больше максимально возможного размера выполнен не будет.