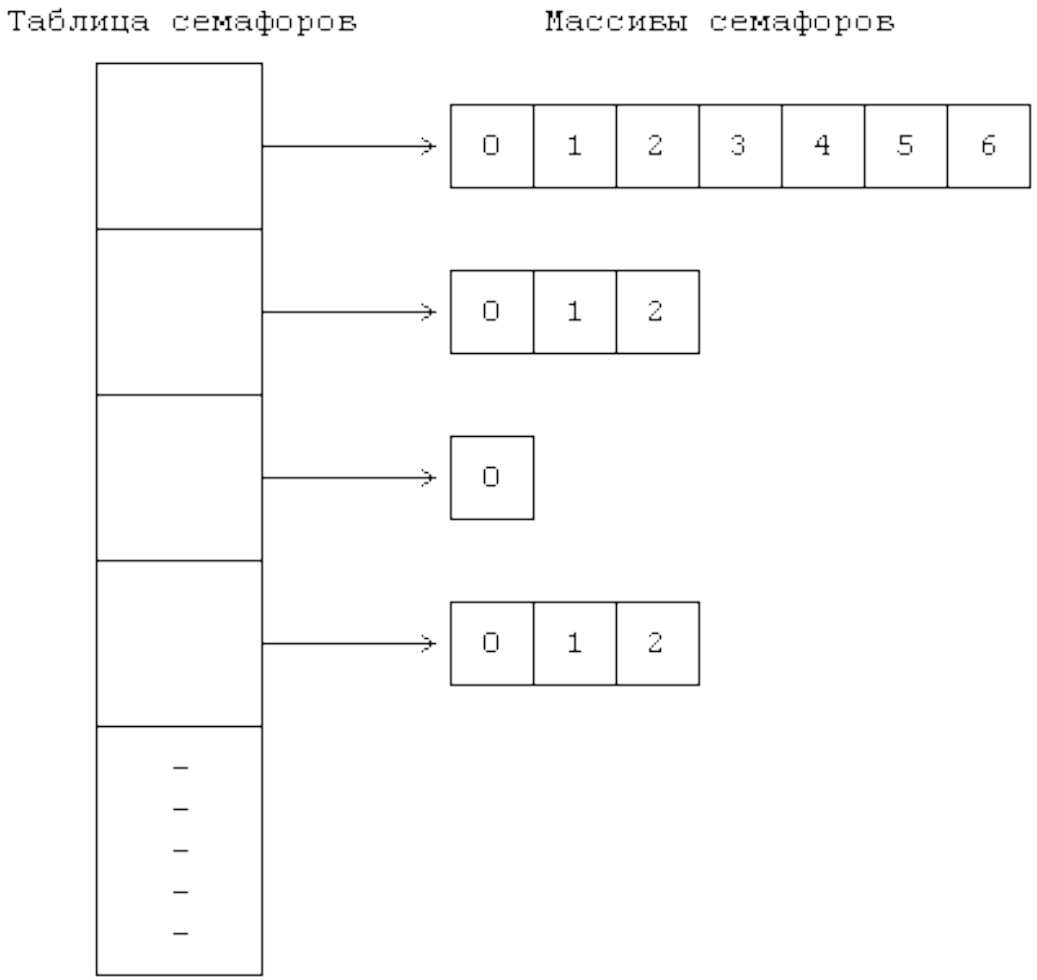
Архитектура ОС UNIX

Санкт-Петербургский государственный политехнический университет

17 ноября 2011

Семафоры

Структуры данных, используемые в работе над семафорами:



Операции установки и снятия блокировки:

```
#define SEMKEY     75
int main() {
struct sembuf psembuf,vsembuf;
int i, first = 1, second = 0, semid, count;
short initarray[2], outarray[2];
semid = semget(SEMKEY,2,0777 | IPC_CREAT);
initarray[0] = initarray[1] = 1;
semctl(semid,2,SETALL,initarray);
semctl(semid,2,GETALL,outarray);
        psembuf.sem_op = -1;
        psembuf.sem_flg = SEM_UNDO;
        vsembuf.sem_op = 1;
        vsembuf.sem_flg = SEM_UNDO;
```

Семафоры

```
for (count = 0; ; count++)
{
psembuf.sem_num = first;
semop(semid,&psembuf,1);
psembuf.sem_num = second;
semop(semid,&psembuf,1);
vsembuf.sem_num = second;
semop(semid,&vsembuf,1);
vsembuf.sem_num = first;
semop(semid,&vsembuf,1);
semctl(semid,2,IPC_RMID,0);
}
```

Очереди сообщений

С сообщениями работают четыре системных функции:

- msgget — возвращает дескриптор определящий очередь сообщений
- msgctl — устанавливает параметры очереди
- msgsnd — посылает сообщение
- msgrcv — получает сообщение

Очереди сообщений

Синтаксис вызова функции:

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

int msgget(key_t key, int msgflg);
```

key — числовой ключ представляющий собой идентификатор записи, задается пользователем, msgflg — флаг который может принимать одно из следующих значений: IPC_PRIVATE, IPC_CREAT, IPC_EXCL. Функция возвращает числовой дескриптор созданной очереди.

Очереди сообщений

Ядро хранит сообщения в связном списке, определяемом значением дескриптора, и использует значение msgqid в качестве указателя на массив заголовков очередей. Содержит следующие поля:

- Указатели на первое и последнее сообщение в списке;
- Количество сообщений и общий объем информации в списке в байтах;
- Максимальная емкость списка в байтах;
- Идентификаторы процессов, пославших и принявших сообщения последними;
- Поля, указывающие время последнего выполнения функций msgsnd, msgrcv и msgctl.

Очереди сообщений

Для посылки сообщения процесс использует системную функцию msgsnd:

```
int msgsnd(int msqid, const void *msgp, size_t
    msgsz, int msgflg);
```

Где msgqid — дескриптор очереди сообщений, msgp — указатель на структуру, состоящую из типа в виде назначаемого пользователем целого числа и массива символов, msgsz — размер информационного массива, msgflag — действие, предпринимаемое ядром в случае переполнения внутреннего буферного пространства.

Алгоритм msgsnd

Входная информация:

1. дескриптор очереди сообщений
2. адрес структуры сообщения
3. размер сообщения
4. флаги

Выходная информация: количество посланных байт.

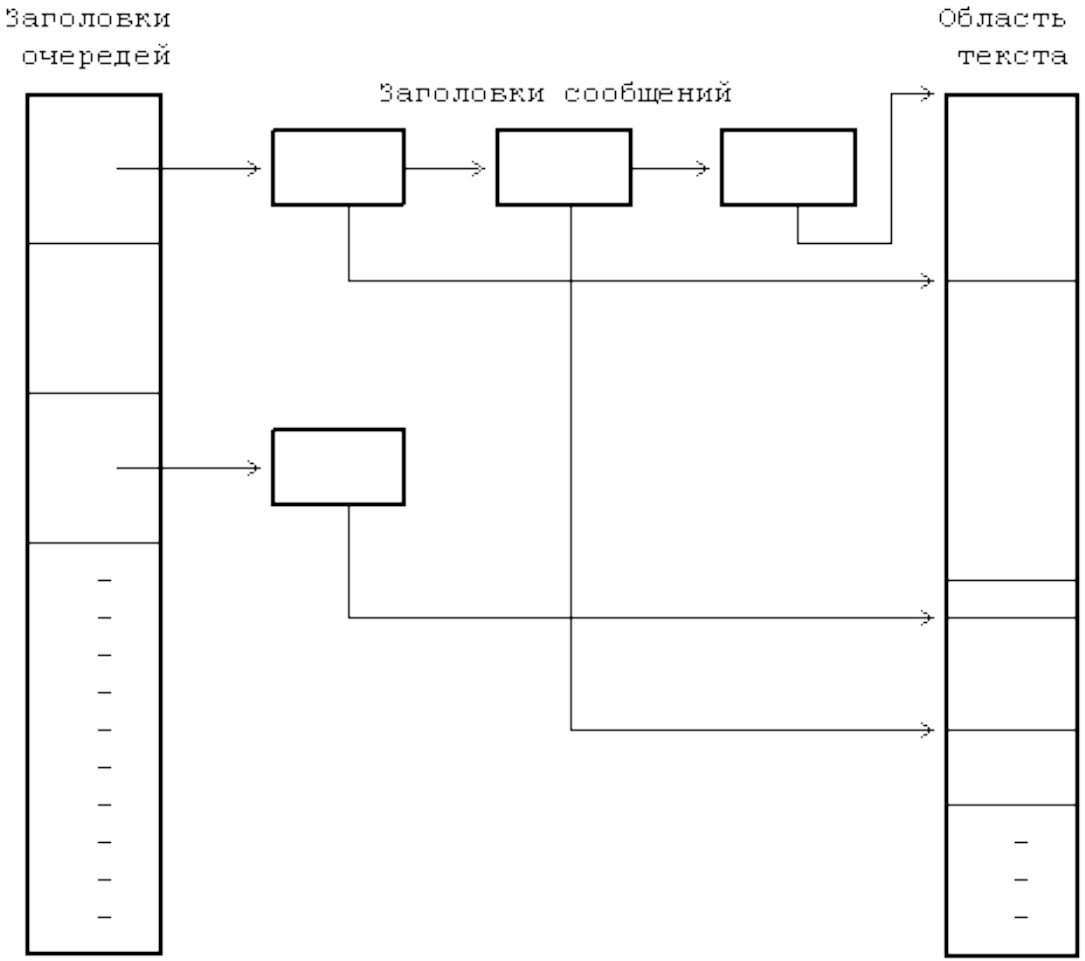
Алгоритм msgsnd

```
{ проверить правильность указания дескриптора и
    наличие соответствующих прав доступа
  выполнить пока (для хранения сообщения не будет
      выделено место)
  { если (флаги не разрешают ждать)
            вернуться
       приостановиться (до тех пор, пока место не
          освободится) }
  получить заголовок сообщения;
  считать текст сообщения из пространства задачи в
       пространство ядра
  настроить структуры данных: выстроить очередь
      заголовков сообщений, установить в заголовке
       указатель на текст сообщения, заполнить
      поля, содержащие счетчики, время последнего
      выполнения операций и идентификатор процесса
  вывести из состояния приостанова все процессы,
      ожидающие разрешения считать сообщение из
      очереди }
```

Алгоритм msgsnd

Ядро проверяет имеется ли у посылающего сообщение процесса разрешения на запись по указанному дескриптору, не выходит ли размер сообщения за установленную системой границу, не содержится ли в очереди слишком большой объем информации, а также является ли тип сообщения положительным целым числом.

Структура данных



Очереди сообщений

Процесс получает сообщения, вызывая функцию msgrcv по следующему формату:

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

Где msgid — дескриптор сообщения, msgp — адрес пользовательской структуры, которая будет содержать полученное сообщение, msgsz — размер структуры msgp, msgtype — тип считываемого сообщения, flag — действие, предпринимаемое ядром в том случае, если в очереди сообщений нет, пользователю возвращается число прочитанных байт сообщения.

Алгоритм msgrcv

Входная информация:

- дескриптор сообщения
- адрес массива, в который заносится сообщение
- размер массива
- тип сообщения в запросе
- флаги

Выходная информация: количество байт в полученном сообщении

Алгоритм msgrcv

```
{       проверить права доступа;
    loop:
        проверить правильность дескриптора сообщения;
        если (тип сообщения в запросе == 0)
            рассмотреть первое сообщение в очереди;
        в противном случае если (тип сообщения в запросе > 0)
            рассмотреть первое сообщение в очереди, имеющее данный тип;
        в противном случае
            рассмотреть первое из сообщений в очереди с наименьшим
                значением типа при условии, что его тип не превышает
                абсолютное значение типа, указанного в запросе;
        если (сообщение найдено)
        { переустановить размер сообщения или вернуть ошибку, если размер,
            указанный пользователем слишком мал;
          скопировать тип сообщения и его текст из пространства ядра в
              пространство задачи;
          разорвать связь сообщения с очередью;
          вернуть управление; }
        если (флаги не разрешают приостанавливать работу)
            вернуть управление с ошибкой;
        приостановиться (пока сообщение не появится в очереди);
        перейти на loop; }
```

Алгоритм msgsnd

```
#define MSGKEY      75
struct msgform
{ long      mtype;
  char      mtext[256]; } msg;
int main() {
        int i,pid,*pint;
        int msgid;
        msgid = msgget(MSGKEY,0777 | IPC_CREAT);
        for (;;)
        {
                msgrcv(msgid,&msg,256,1,0);
                pint = (int *) msg.mtext;
                pid = *pint;
                msg.mtype = pid;
                *pint = getpid();
                msgsnd(msgid,&msg,sizeof(int),0);
        }
        msgctl(msgid,IPC_RMID,0);
}
```

Очереди сообщений

С помощью системной функции msgctl процесс может запросить информацию о статусе дескриптора сообщения, установить этот статус или удалить дескриптор сообщения из системы. Синтаксис вызова функции:

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf
    );
```

Где msgid — дескриптор сообщения, cmd — тип команды, buf — адрес пользовательской структуры, в которой будут храниться управляющие параметры или результаты обработки запроса.

Разделяемая память

Процессы могут взаимодействовать друг с другом непосредственно путем совместного использования участков виртуального адресного пространства и обмена данными через разделяемую память.
Для работы с разделяемой памятью используюся следующие функции:

- shmget — создает новую область разделяемой памяти или возвращает адрес уже существующей области
- shmat — логически присоединяет область к виртуальному адресному пространству процесса
- shmdt — отсоединяет область от виртуального адресного пространства
- shmctl — устанавливает различные параметры, связанными с разделяемой памятью.

Разделяемая память

Синтаксис вызова системной функции shmget:

- shmget — создает новую область разделяемой памяти или возвращает адрес уже существующей области
- shmat — логически присоединяет область к виртуальному адресному пространству процесса
- shmdt — отсоединяет область от виртуального адресного пространства
- shmctl — устанавливает различные параметры, связанными с разделяемой памятью.

Разделяемая память

Синтаксис вызова системной функции shmget:

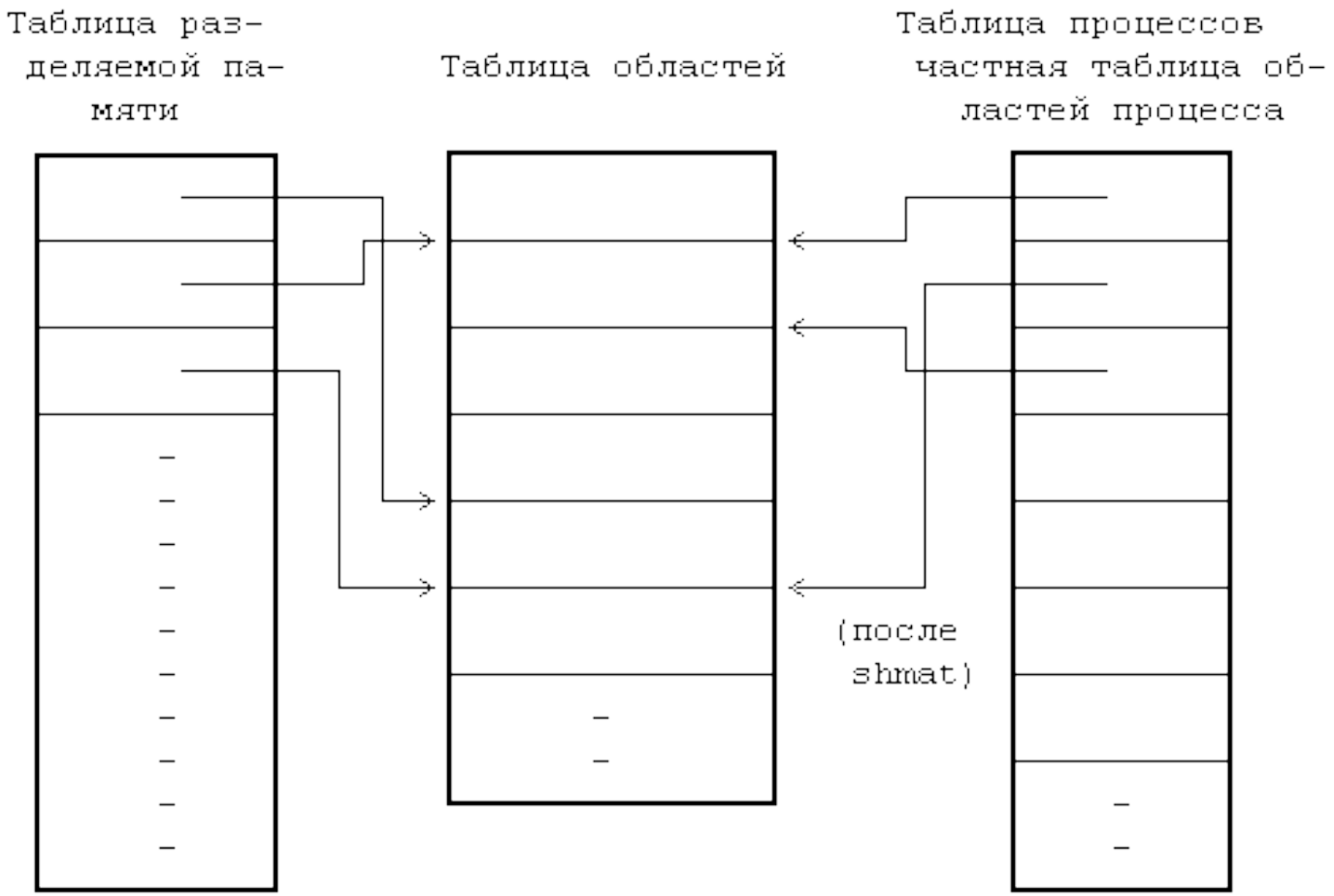
```
#include <sys/ipc.h>
#include <sys/shm.h>

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

Где size — объем области в байтах. Ядро использует key для ведения поиска в таблице разделяемой памяти: если подходящая запись обнаружена и если разрешение на доступ имеется, ядро возвращает вызывающему процессу указанный в записи дескриптор.

Разделяемая память

Если запись не найдена и если пользователь установил флаг IPC_CREAT, указывающий на необходимость создания новой области, ядро проверяет нахождение размера области в установленных системой пределах и выделяет область по алгоритму allocreg. Ядро записывает установки прав доступа, размер области и указатель на соответствующую запись таблицы областей в таблицу разделяемой памяти и устанавливает флаг, свидетельствующий о том, что с областью не связана отдельная память.

Разделяемая память



Разделяемая память

Процесс присоединяет область разделяемой памяти к своему виртуальному адресному пространству с помощью системной функции shmat:

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

shmid — идентифицирует область разделяемой памяти, shmaddr — является виртуальным адресом, по которому пользователь хочет подключить область, если пользователь хочет чтобы ядро выбрало автоматически то необходимо передать NULL, shmflags — указывает предназначена ли область только для чтения. Возвращаемое функцией значение, представляет собой виртуальный адрес, по которому ядро произвело подключение области и который не всегда совпадает с адресом, указанным пользователем.

Алгоритм shmat

Входная информация:

1. дескриптор области разделяемой памяти
2. виртуальный адрес для подключения области
3. флаги

Выходная информация: виртуальный адрес, по которому область подключена фактически.

Алгоритм shmat

```
{
    проверить правильность указания дескриптора,
        права доступа к области;
    если (пользователь указал виртуальный адрес)
    { округлить виртуальный адрес в соответствии с
        флагами;
            проверить существование полученного адреса
                , размер области; }
    в противном случае
            ядро выбирает адрес: в случае неудачи
                выдается ошибка;
    присоединить область к адресному пространству
        процесса (алгоритм attachreg);
    если (область присоединяется впервые)
            выделить таблицы страниц и отвести память
                под нее (алгоритм growreg);
    вернуть (виртуальный адрес фактического
        присоединения области);
}
```

Разделяемая память

Отсоединение области разделяемой памяти от виртуального адресного пространства процесса выполняется функцией:

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

Где shmaddr - виртуальный адрес, возвращенный функцией shmat, используя алгоритм detachreg. Возвращает 0 в случае успешного освобождения памяти и -1 в случае ошибки.

Разделяемая память

Процесс запрашивает информацию о состоянии области разделяемой памяти и производит установку параметров для нее с помощью системной функции shmctl:

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

shmid — идентифицирует запись таблицы разделяемой памяти, cmd — определяет тип операции, buf — является адресом пользовательской структуры, в которую помещается информация о состоянии области.

Алгоритм shmat

Присоединение процессом одной и той же области разделяемой памяти дважды:

```
#define SHMKEY     75
int main() {
int shmid, i, *pint;
char *addr1, *addr2;
shmid = shmget(SHMKEY,128*1024,0777 | IPC_CREAT);
addr1 = shmat(shmid,0,0);
addr2 = shmat(shmid,0,0);
printf("addr1 0x%x addr2 0x%x\n",addr1,addr2);
pint = (int *) addr1;
for (i = 0; i < 256, i++)
*pint++ = i;
pint = (int *) addr1;
*pint = 256;
pint = (int *) addr2;
for (i = 0; i < 256, i++)
printf("index %d\tvalue %d\n",i,*pint++);
shmctl(shmid,IPC_RMID,0);
```

Алгоритм shmat

Разделение памяти между процессами:

```
//Master process 1 (create SHM)
#define SHMKEY     75
#define     K     1024
int main()
{
int shmid;
int i, *pint;
char *addr;
shmid = shmget(SHMKEY,64*K,0777| IPC_CREAT);
addr = shmat(shmid,0,0);
pint = (int *) addr;
while (*pint == 0);
for (i = 0; i < 256; i++)
printf("%d\n",*pint++);
}
```

Алгоритм shmat

Разделение памяти между процессами:

```
//Client process 2
int main()
{
int shmid;
int i, *pint;
char *addr;
shmid = shmget(SHMKEY,64*K,0777);
addr = shmat(shmid,0,0);
pint = (int *) addr;
for (i = 1; i <= 256; i++)
*pint++ = i;

shmdt(addr); //detach memory
shmctl(shmid, IPC_RMID, 0); //remove shared memory
    key
}
```