UNIX

Лекция 8

Глобальные таблицы

- Семафоры, разделяемая память, сообщения
- С каждым механизмом связана таблица, в записях которой описываются все его детали.
- В каждой записи содержится числовой ключ (***key***) - идентификатор записи, выбранный пользователем. Ключ – это аналог имени файла. Имя важно лишь для пользователя. Систему интересует **дескриптор** записи.

Ключ KEY (номер записи)	Дескриптор	Прочая информация
1	101	…
2	102	…
…	…	…
…	…	…

Глобальные таблицы

- В каждом механизме имеется системная функция типа «***get***», используемая для создания новой или поиска существующей записи; параметры функции - **идентификатор** записи и различные флаги.
- Ядро ищет запись по ее идентификатору.
- IPC_PRIVATE - получение еще неиспользуемой записи.
- IPC_CREAT - создание новой записи, если записи с указанным идентификатором нет, а если еще к тому же установить флаг IPC_EXCL, можно получить уведомление об ошибке, если запись с таким идентификатором существует.
- Функция возвращает некий выбранный ядром дескриптор
- Аналог функций creat и open.

Ключ KEY (номер записи)	Дескриптор	Прочая информация
1	101	…
2	102	…
…	…	…
…	…	…

Глобальные таблицы. Дескрипторы

- В каждой записи таблицы содержится значение дескриптора записи. Поиск по дескриптору указателя на запись в таблице структур данных:

  n="<значение дескриптора> mod <число записей в таблице>"
- Когда процесс удаляет (освобождает) запись, ядро увеличивает значение связанного с ней дескриптора на число записей в таблице (D=D+N).
- Процессы, которые будут пытаться обратиться к записи по ее старому дескриптору, потерпят неудачу.

Ключ KEY (номер записи)	Дескриптор	Прочая информация
1	101	…
2	102	…
…	…	…
…	…	…

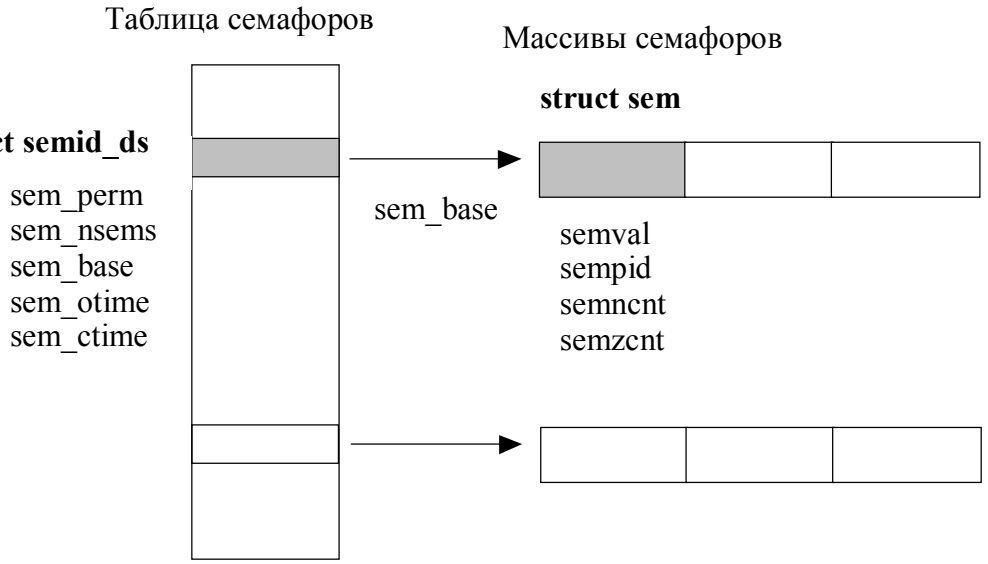
Функция *ftok*

key_t ftok(char *path, char key)

- Функция по заданному имени файла и значению параметра *key* (трактуемому как номер версии проекта) выдает число целое число, могущее использоваться в качестве значения ключа.
- Файл, имя которого указано аргументом *path*, должен существовать.
- При этом манипулировать этим файлом не следует, т.к. для вычисления значения ключа эта функция использует номер *inode* этого файла.

СЕМАФОРЫ UNIX

- Таблица семафоров - глобальная структура.
- В записях таблицы описываются структуры **semid_ds**.
- Каждая запись имеет имя - целочисленный ключ.
- Каждая запись содержит структуру данных, описывающую права доступа к ней, а также управляющую информацию.
- Каждый семафор представляет собой набор значений (**вектор** семафоров).
- Одновременное выполнение нескольких операций (векторные операции).



Семафор

- значение семафора,
- идентификатор последнего из процессов, работавших с семафором,
- количество процессов, ожидающих увеличения значения семафора,
- количество процессов, ожидающих момента, когда значение семафора станет равным 0.

```
struct sem
{   short semval;          // текущее значение семафора
    short sempid;          // pid процесса, который
                           // выполнял последним операции
                           // над этим семафором
    short semncnt;         // число процессов, которые заблокированы
                           // и ожидают увеличения значения семафора
    short semzcnt;         // число процессов, которые заблокированы
                           // и ожидают обнуления значения семафора
}
```

Таблица семафоров

```
struct semid_ds
{     struct          ipc_perm sem_perm;        //права доступа
      time_t          sem_otime;        //время выполнения
                                        //последней операции (semop)
      time_t          sem_ctime;        //время изменения
                                        //управляющих параметров (semctl)
      struct sem      *sem_base;        //указатель на первый семафор
                                        // в массиве
      struct sem_undo *undo;            //указатель на
                                        // структуру восстановления
      ushort          sem_nsems;        //количество семафоров в массиве
}
```

Основные особенности семафоров UNIX SystemV:

- Глобальный характер семафоров (в отличие от семафоров POSIX);
- Семафоры представляют собой массив, или множество атомарных семафоров;
- Векторный характер операций: можно производить операции над всеми элементами набора семафоров одновременно. Если хоть одна операция является блокирующей, то выполнение всего вектора операций откладывается.

Системные вызовы для работы с семафорами

SEMGET. Создание набора семафоров и получение доступа к ним:

int semget(key_t key, int count, int semflg)

- *key* - номер семафора,
- *count* - количество семафоров,
- *semflg* - параметры создания и права доступа.
- Ядро использует *key* для ведения поиска в таблице семафоров: если подходящая запись обнаружена и разрешение на доступ имеется, ядро возвращает вызывающему процессу указанный в записи **дескриптор**. Если запись не найдена, а пользователь установил флаг IPC_CREAT, - ядро проверяет возможность его создания и выделяет запись в таблице семафоров. Функция возвращает номер дескриптора набора семафоров.

Примеры:

1. Создание набора семафоров из 5 штук:

- *fd = semget(100, 5, IPC_CREAT|0666);*
- IPC_PRIVATE - будет создан уникальный семафор, доступный только родственным процессам
- IPC_EXCL - если такой семафор уже есть, то fd =-1;

2. Получение доступа к семафору, который уже есть:

- *fd = semget(100, 0, 0)*

SEMOP. Установка или проверка значения семафора

*int semop(int semid, struct sembuf *oplist, unsigned nsops)*

- *semid* - дескриптор, возвращаемый функцией semget
- oplist - указатель на список операций
- nsops - размер списка.
- Возвращаемое функцией значение является прежним значением семафора, над которым производилась операция.

```
struct sembuf
{   unsigned short sem_num;     //индекс семафора в массиве
    short sem_op;           //код операции над семафором
    short sem_flg;          //флаг операции IPC_NOWAIT, SEM_UNDO
}
```

Операции выполняются комплексно - или все за один сеанс, или ни одной.

SEMOP

sembuf*:: sem_flg* – флаги операции

- IPC_NOWAIT: если ядро попадает в такую ситуацию, когда процесс должен приостановить свое выполнение в ожидании увеличения значения семафора выше определенного уровня или, наоборот, снижения этого значения до 0, и если при этом флаг IPC_NOWAIT установлен, ядро выходит из функции с извещением об ошибке.
- SEM_UNDO позволяет избежать блокирования семафора процессом, который закончил свою работу прежде, чем освободил захваченный им семафор. Если процесс установил флаг SEM_UNDO, то при завершении этого процесса ядро даст обратный ход всем операциям, выполненным процессом.

SEMOP

sembuf*::* short sem_op - код операции над семафором

- sem_op>0: ядро увеличивает значение семафора и выводит из состояния приостанова все процессы, ожидающие наступления этого события.
- sem_op=0: ядро проверяет значение семафора: если оно равно 0, процесс переходит к выполнению других операций; иначе ядро увеличивает число приостановленных процессов, ожидающих, когда значение семафора станет нулевым, и процесс «засыпает».

struct sem: short semval;

- sem_op<0 или |sem_op|<semval, то semval = semval + sem_op.
- Если semval == 0, ядро выводит из состояния приостанова все процессы, ожидающие обнуления значения семафора.
- Если semval< |sem_op|, ядро приостанавливает процесс до тех пор, пока значение семафора не увеличится.

SEMOP

Примеры: Аналог функции ***P***:

```
struct sembuf sb= {3, -1, SEM_UNDO};
n = semop(fd, &sb, 1);        // если n равно -1, то это
                              // значит, что операция неуспешна
```

Векторная операция над тремя элементами набора:

```
struct sembuf sb[3]=
{   0, 1, SEM_UNDO,//операция с нулевым семафором
    3, -1, SEM_UNDO,//операция с третьим семафором
    10, 1, SEM_UNDO};//операция с десятым семафором

semop(fd, sb, 3);     //выполнили векторную операцию
```

SEMCTL. Выполнение управляющих операций над набором семафоров

int semctl(int semid, int semnum, int cmd, union semun arg)

```
union semun
{     int val;                  // используется только для SETVAL
      struct semid_ds *buf;     // для IPC_STAT и IPC_SET
      unsigned short *array;
} arg;
```

Ядро интерпретирует параметр arg в зависимости от значения параметра cmd, который может принимать следующие значения:

- GETVAL - вернуть значение того семафора, на который указывает параметр semnum.
- SETVAL - установить значение семафора, на который указывает параметр semnum, равным значению arg.val.
- GETPID - вернуть идентификатор процесса, выполнявшего последним функцию semop по отношению к тому семафору, на который указывает параметр semnum.
- GETNCNT - вернуть число процессов, ожидающих того момента, когда значение семафора станет положительным.
- GETZCNT - вернуть число процессов, ожидающих того момента, когда значение семафора станет нулевым.
- GETALL - вернуть значения всех семафоров в массиве arg.array.
- SETALL - установить значения всех семафоров в соответствии с содержимым массива arg.array.
- IPC_STAT - считать структуру заголовка семафора с идентификатором id в буфер arg.buf. Аргумент semnum игнорируется.
- IPC_SET - запись структуры семафора из буфера arg.buf.
- IPC_RMID - удалить семафоры, связанные с идентификатором id, из системы.

SEMCTL

int semctl(int semid, int semnum, int cmd, union semun arg)

```
union semun
{     int val;                  // используется только для SETVAL
      struct semid_ds *buf;// для IPC_STAT и IPC_SET
      unsigned short *array;
} arg;
```

Cmd	val	buf	array	semnum
IPC_STAT	-	+	-	-
IPC_SET	-	+	-	-
IPC_RMID	-	-	-	-
GETALL	-	-	+	-
SETALL	-	-	+	-
GETVAL	-	-	-	+
SETVAL	+	-	-	+
GETPID	-	-	-	+
GETNCNT	-	-	-	+
GETZCNT	-	-	-	+

Пример. LINUX CALC SERVER/CLIENT HEADER

```
1.  #ifndef _SC_H_
2.  #define _SC_H_
3.  #include <sys/ipc.h>
4.  #include <sys/shm.h>
5.  #include <sys/sem.h>
6.  /* shared memory key */
7.  #define SHMid 123
8.  typedef struct
9.  {       int data;
            char string[512];
            int count;
            int cmd;
10. } SHMdata;
11. #define SEMid 321
12. /* semaphores */
13. #define access      0
14. #define answer      1
15. #define query       2
16. /* semaphore operator */
17. struct sembuf sop;
18. int sem_wait(int sid, int snum)
19. /* function P */
20. {       sop.sem_num = snum; /* semaphore number */
21.         sop.sem_op = -1;
22.         return semop(sid,&sop,1);
23. }
24. int sem_post(int sid, int snum)
25. /* function V */
26. {       sop.sem_num = snum; /* semaphore number */
27.         sop.sem_op = 1;
28.         return semop(sid,&sop,1);
29. }
30. #endif
```

LINUX CALC SERVER

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <fcntl.h>
4.  #include <errno.h>
5.  #include <stdlib.h>
6.  #include <sys/mman.h>
7.  #include <sys/types.h>
8.  #include <sys/ipc.h>
9.  #include <sys/shm.h>
10. #include "sc.h"
11. SHMdata *addr;
12. void serror(char *msg)
13. { printf("\nserver error [%d]: %s:%s\n",getpid(),msg,strerror(errno));
14.   exit(1);
15. }
16. void main(int argc, char *argv[])
17. { int nval;
18.   int sid;
19.   int shmid;
20.   short initarray[3] = {1,0,0}; /* access,answer,query */
21.   /* Open sh memory */
22.   shmid = shmget(SHMid,sizeof(SHMdata), 0777 | IPC_CREAT);
23.   addr = (SHMdata *)shmat(shmid,0,0);
24.   printf("\nserver: map address = %6.6X\n",addr);
25.   /* create and init semaphores */
26.   sid = semget(SEMid,3,0777 | IPC_CREAT);
27.   /* init semaphores*/
28.   semctl(sid,3,SETALL,initarray);
29.   /* access=1,answer=0,query=0 */
30.   // sem_init(&SEM->access,1,1)
31.   // sem_init(&SEM->answer,1,0)
32.   // sem_init(&SEM->query,1,0)
33.   /* main program loop */
34.   addr->cmd = 1;
      while(1)
      { if(sem_wait(sid,query)<0)
          serror("sem_wait/query");
        if(addr->cmd==0) break;
        nval = (addr->data)+100;
        (addr->data) = nval;
        if(sem_post(sid,answer)<0)
          serror("sem_post/answer");
        if(sem_post(sid,access)<0)
          serror("sem_post/access");
      }
      /* delete semaphore */
      semctl(sid,3,IPC_RMID,0);
      /* delete shared memory*/
      shmctl(shmid,IPC_RMID,0);
      printf("\n\nserver terminated\n");
}
```

```
typedef struct
{  int data;
          char
        string[512];
          int count;
          int cmd;
} SHMdata
```

LINUX CALC CLIENT

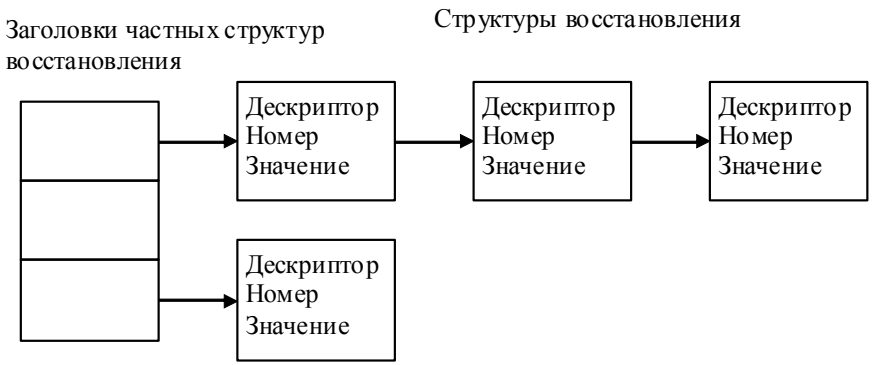
```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "sc.h"
SHMdata      *addr;
void serror(char *msg)
{       printf("\nclient error [%d]:
        %s:%s\n",getpid(),msg,strerror(errno));
        exit(1);
}
void main(int argc, char *argv[])
{       int shmid;
        int semid;
        int counter, nval;
        /* Open sh memory */
        shmid = shmget(SHMid, sizeof(SHMdata), 0);
        addr = (SHMdata *)shmat(shmid,0,0);
        printf("\nclient [%d]: map address =
        %6.6X\n",getpid(),addr);
        /* open semaphores */
        semid = semget(SEMid,0,0);
```

```
typedef struct
{  int data;
           char
        string[512];
           int count;
           int cmd;
} SHMdata
```

```
        /* main program loop */
        for(counter=1;counter<argc;counter++)
        { sleep(1);
          /* wait for access */
          if(sem_wait(semid,access)<0)
            serror("sem_wait/access");
          nval = atoi(argv[counter]);
          printf("\n\tclient [%d]: -Count me
                  %d",getpid(),nval);
          addr->data=nval;
          addr->cmd = nval;
          sem_post(semid,query);
          /* wait for answer */
          if(sem_wait(semid,answer)<0)
            serror("sem_wait/access");
          printf("\n\tclient [%d]: -Got answer %d",
                   getpid(),addr->data);
        }
        printf("\n** client [%d]
        terminated\n",getpid());
}
```

Структуры восстановления

- Если процесс выполняет операцию над семафором и завершает свою работу без приведения семафора в исходное состояние, могут возникнуть опасные ситуации.
- В *semop()* процесс может установить флаг операции SEM_UNDO: при завершении процесса ядро даст обратный ход всем операциям, выполненным этим процессом над семафорами.




```
struct sem_undo
{  struct sem_undo *proc_next; // ссылка на
        // след. элемент для этого процесса
  struct sem_undo *id_next;     // ссылка на
           // след. элемент для этого набора
  int  semid;  // идентификатор массива
                  // семафоров
  short  *semadj;  // массив значений каждого
                        // из семафоров
}
```

Работа со структурами восстановления (СВ)

- Для восстановления необходимо иметь информацию об ID семафора и значениях элементов набора.
- Каждая СВ содержит ID семафора, его порядковый номер в наборе и установочное значение.
- Ядро выделяет СВ динамически, во время первого выполнения *semop* с установленным флагом **SEM_UNDO**.
- При последующих обращениях к функции с тем же флагом ядро просматривает СВ для процесса в поисках структуры с тем же самым ID и порядковым номером семафора, что и в вызове функции.
- Если СВ обнаружена, ядро вычитает значение произведенной над семафором операции из установочного значения. Т.о., в СВ хранится результат вычитания суммы значений всех операций, произведенных над семафором, для которого установлен флаг SEM_UNDO.
- Если соответствующей СВ нет, ядро создает ее, сортируя при этом список СВ по ID и номерам семафоров. Если установочное значение становится равным 0, ядро удаляет СВ из списка. Когда процесс завершается, ядро вызывает процедуру, которая просматривает все связанные с процессом СВ и выполняет над указанным семафором все обусловленные действия.
- Ядро создает СВ всякий раз, когда процесс уменьшает значение семафора, а удаляет ее, когда процесс увеличивает значение семафора, поскольку установочное значение структуры равно нулю.