UNIX

Лекция 7

СЕМАФОРЫ И РАЗДЕЛЯЕМАЯ ПАМЯТЬ

Проблема конкуренции за общие ресурсы.

Если по условиям работы требуется, чтобы разделяемые ресурсы одномоментно были доступны только одному процессу, то такие ресурсы называются ***критическими***.

Процесс упорядочения общения между конкурирующими процессами называется ***синхронизацией****.*

Способы синхронизации

Системы синхронизации процессов:

- блокировка памяти;
- **семафоры**;
- критические области;
- условные критические области;
- мониторы;
- исключающие области и т.д.

Семафоры

*Примитивы **P** и **V***

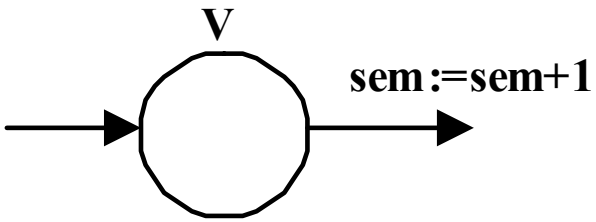
Эдсгер Дейкстра, 1965 г.

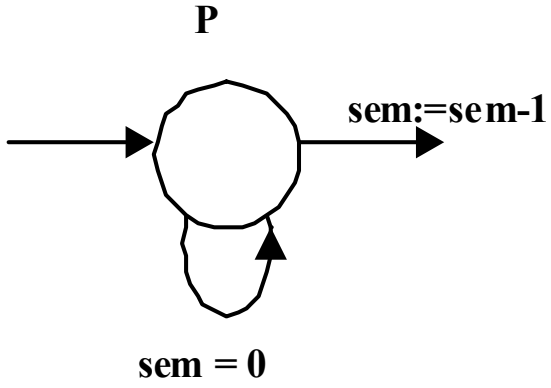
*"passeren" (**P**) – занять, закрыть*

*"vrijgeven" (**V**) – освободить.*

V - операция (V(S)): *операция с одним аргументом, который должен быть семафором. Эта операция увеличивает значение аргумента на 1.*

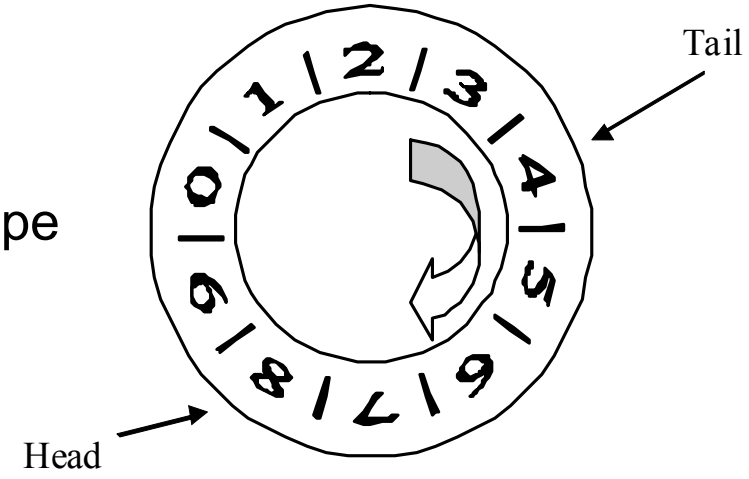
P - операция (P(S)): операция с одним аргументом, который должен быть семафором. Ее назначение – уменьшить величину аргумента на 1, если только результирующее значение не становится отрицательным.





Задача о кольцевом буфере

Буфер на N ячеек
Голова (Head) и хвост (Tail)
Head – первая свободная ячейка,
Tail – последняя занятая.
Пишем в голову, читаем из хвоста.



Семафоры:
1. empty - количество данных в буфере
2. full - количество свободных мест
3. fetchprotect
4. storeprotect - вспомогательные семафоры, обеспечивающие взаимоисключение

Семафоры и разделяемая память стандарта POSIX

Стандарт POSIX (Portable Operating System Interface for Computing Environment) определяет то, как должны выглядеть системные вызовы.

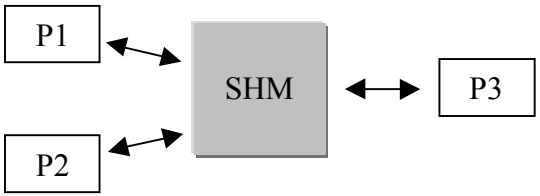
Некоторые полностью POSIX-совместимые системы

- Полностью соответствующие одной из версий стандарта POSIX.
- HP-UX
- IBM AIX
- IRIX
- LynxOS
- Mac OS X
- Minix
- OpenSolaris
- QNX
- Solaris

Частично POSIX-совместимые

- FreeBSD
- Linux
- OpenBSD
- Sanos

Разделяемая память стандарта POSIX



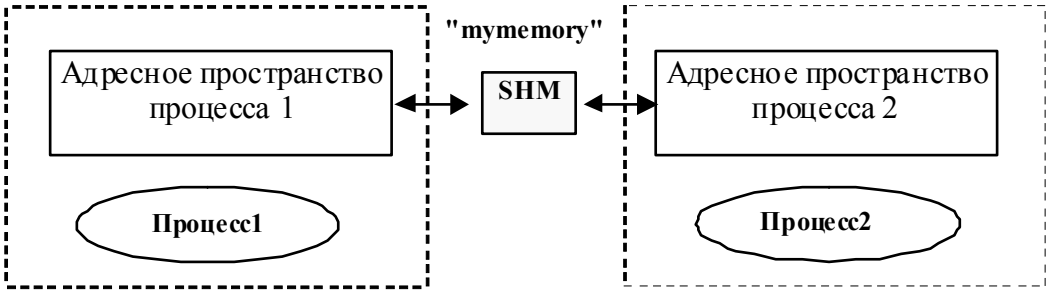
int ***shm_open****(char *name, int flags, mode_t mode)*

- *name* – путевое имя области;
- *flags, mode* – режим открытия и атрибуты области разделяемой памяти (полностью аналогичны соответствующим параметрам open).

int ***ftruncate****(int fd, off_t shared_memory_size) или*
long ***ltrunc****(int fd, long offset, int fromwhere)*

shm_unlink*(char *name)*
Удаление области разделяемой памяти

Пример



Процесс 1	**Процесс 2**
shm_open(“mymemory”, O_CREATE| O_TRUNC | O_RDWR, 0777); ltrunc(f, 1024,0); …. shm_unlink(“mymemory”);	shm_open(“mymemory”, O_RDWR) … shm_unlink(“mymemory”);

После выделения области памяти и определения ее размера необходимо вызвать функцию ***mmap***, которая отобразит разделяемую область памяти в виртуальное адресное пространство вызывающего процесса.

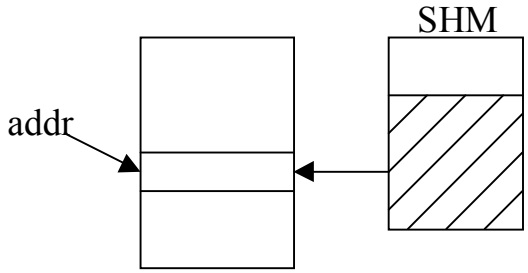
Ввод-вывод с отображением в память

<sys/mman.h>.

*1. caddr_t **mmap**(caddr_t addr, int size, int prot, int flags, int fd, off_t pos)*

- отображение файлового объекта в виртуальное адресное пространство процесса. Возвращает адрес, по которому подсоединилась разделяемая память.
- *addr* - адрес начала отображения. Лучше установить его в 0, тогда сама система определит, по какому адресу присоединить память;
- *size* - размер отображаемой памяти
- *prot* - права доступа к отображаемой памяти (PROT_READ, PROT_WRITE, PROT_EXEC). Указывает, что будем делать с памятью: только читать, только записывать или выполнять;
- *flags* - опции отображения (MAP_SHARED, MAP_PRIVATE, MAP_FIXED). Показывает кому принадлежит память: если он равен MAR_SHARED, то разделяемая память общедоступна, если он равен MAP_PRIVATE, то это – собственность процесса; MAP_FIXED – флаг, требующий выделения памяти, начиная точно с адреса *addr*.
- *fd* - файловый дескриптор;
- *pos* - начальная позиция в файловом объекте. Обычно этот параметр равен 0 (либо его значение должно быть кратным размеру страницы памяти).

Отсоединение



munmap*(caddr_t addr, int size)*

Пример программы

Программа, копирующая файл с использованием отображения файла в память:

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <sys/mman.h>
4.  #include <sys/unistd.h>
5.  #include <sys/fcntl.h>
6.  main(int argc, char *argv[])
7.  {       int fin, fout;
8.          caddr_t source, dest;
9.          struct stat filestat;
10.         fin = open(argv[1],O_RDONLY);
11.         fout = open(argv[2],O_CREAT|O_TRUNC|O_WRONLY,0666);
12.         fstat(fin,filestat);
13.         lseek(fout,filestat.st_size-1,SEEK_SET);
14.         write(fout,"",1);
15.         source = mmap(0, filestat.st_size,PROT_READ,MAP_SHARED,fin,0);
16.         dest = mmap(0, filestat.st_size, PROT_READ | PROT_WRITE,
17.                                     MAP_SHARED,fout,0);
18.         memcpy(dest,source, filestat.st_size);
19.         exit(0);
20. }
```

Семафоры стандарта POSIX

```
typedef struct
{
  volatile  long value;
   long              semid;
} sem_t;
```

Системные вызовы для работы с семафорами

1. int **sem_init**(sem_t *sem, int pshared, unsigned value)
Создать семафор.

sem – указатель на структуру;

pshared – является ли семафор доступным остальным процессам. Если *pshared* =1, то семафор общедоступен; если значение *pshared=0*, то семафор доступен только процессу и его потомкам.

value – начальное значение семафора.

*2. int **sem_destroy**(sem_t *sem)* Уничтожить семафор

*3. int **sem_close**(sem_t *dem)*
Закрыть семафор (без его удаления – в отличие от функции sem_destroy)

*4. int **sem_post**(sem_t *sem)* Открыть семафор. Аналог функции **V**.

*5. int **sem_wait**(sem_t *sem)* Ждать открытия семафора. Аналог функции **P.**

*6. int **sem_trywait**(sem_t *sem)* Проверка значения переменной семафора

*7. int **sem_getvalue**(sem_t *sem, int *value)* Получить значение переменной семафора

```
typedef struct
{
        volatile  long value;
        long        semid;
} sem_t;
```

Дополнительно

sem_open*(char *name, int flags, mode_t mode, unsigned value)*

sem_unlink*(char *name)*

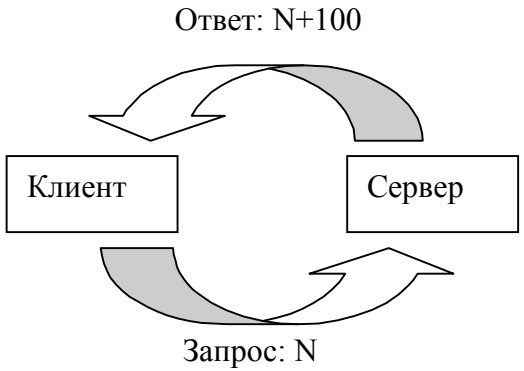
Функция sem_*open*, фактически, создает специальный файл с именем name и содержащим структуру семафора. Аргументы *flags* и *mode* имеют тот же смысл, что и в системном вызове *open*.

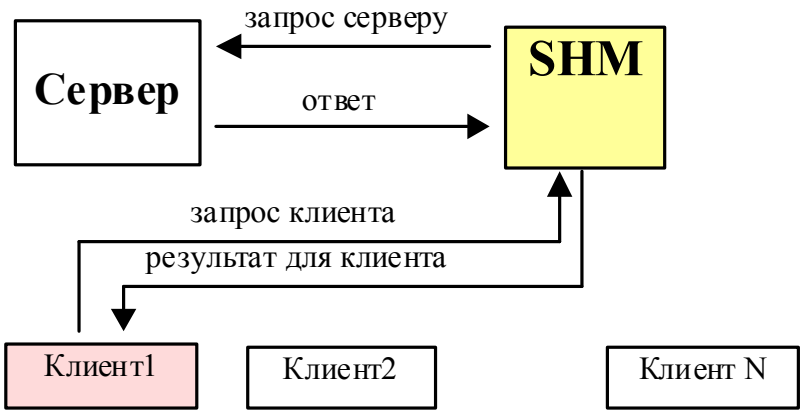
```
1. #include <stdio.h>
2. #include <sys/stat.h>
3. #include <semaphore.h>
4. main()
5. {     sem_t *semp = semopen("/sem.0",O_CREATE, S_IRWXU,1);
6.       if(semp==(sem_t *)-1) perror("…");
7.       …
8.       sem_close(semp);
9.       sem_unlink("/sem.0");
10.}
```

Пример использования семафоров и разделяемой памяти

Задача: написать приложение типа «клиент-сервер».

- Клиентов много (работают параллельно), сервер их обслуживает.
- Клиент посылает запрос серверу - число N, а сервер возвращает клиенту число (N+100).
- Если клиент отправляет число 0, то работа сервера завершается.

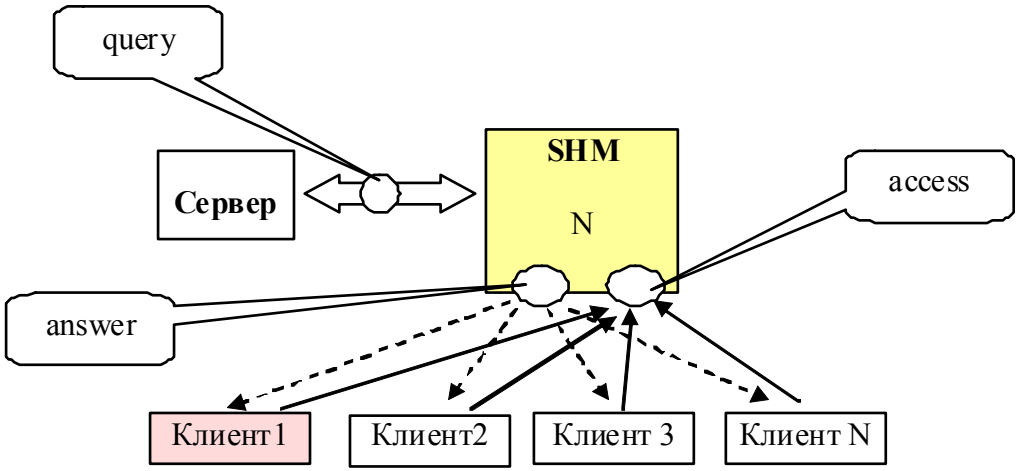




Решение

Потребуются три семафора:

- *access* - семафор, который говорит, занят ли ресурс. Это - семафор **доступа**, который нужен, чтобы контролировать доступ к разделяемой памяти;
- *query* - семафор запроса (необязательный), т.к. если никто ничего не хочет от сервера, то сервер не работает, а ждет. Это - семафор запроса, который нужен, чтобы сервер ждал пока появится запрос.
- *answer* – семафор, заставляющий клиента ждать ответа. Это - семафор ответа, который нужен, чтобы процесс ждал помещения ответа;



Нужно сделать все семафоры **общедоступными** для всех процессов, => используем разделяемую память.

Требуется две области разделяемой памяти:

1. для данных,
2. для семафоров.

SC.H QNX CLIENT/SERVER HEADER FILE. POSIX VERSION

```
1. /* Структура, определяющая данные. Структура запроса совпадает со
   структурой ответа на запрос*/
2. typedef struct
3. {    int data;
4.      int cmd;
5. } SHMdata;
6. /*путевое имя разделяемой памяти; аналогично определению имени
   файла */
7. char *SHMname = "DATAMEMORY";
8. typedef struct
9. {    sem_t access;  // семафор доступа
10.     sem_t answer;  // семафор ответа
11.     sem_t query;   // семафор запроса
12.} SEMAPHORE;
13./*семафор должен быть доступен: объявляемые семафоры должны
   находится в разделяемой памяти*/
14.char *SEMmemory = "SEMMEMORY";
15.void serror(char *msg)
16.{    printf("\nerror [%d]: %s:%s\n",getpid(),msg,
                        strerror(errno));
17.     exit(1);
18.}
```

SERVER

```
1.  /* QNX SERVER. POSIX VERSION */
2.  #include <stdio.h>
3.  #include <string.h>
4.  #include <fcntl.h>
5.  #include <errno.h>
6.  #include <stdlib.h>
7.  #include <sys/mman.h> /*описание функций для работы с разделяемой памятью*/
8.  #include <semaphore.h> /*описание функций семафоров*/
9.  #include "sc.h"
10. SEMAPHORE       *SEM;
11. SHMdata         *addr;
12. void main(int argc, char *argv[])
13. {     int fd, nval;
14. int counter;
15. /* Создаем разделяемую память */
16. fd = shm_open(SHMname, O_RDWR | O_CREAT | O_EXCL, 0777);
17. if(fd<0)
18. {     printf("\ncan't open shmem (%s). Try to close it\n", strerror(errno));
19.       shm_unlink(SHMname);
20.       fd = shm_open(SHMname, O_RDWR | O_CREAT | O_EXCL, 0777);
21.       if(fd<0) puts("can't open shmem");
22. }
23. /* Устанавливаем размер области */
24. if(ltrunc(fd,sizeof(SHMdata),SEEK_SET)==-1) serror("set shm size error");
25. addr = mmap(0,sizeof(SHMdata), PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
26. if(addr == (void *)-1) serror("mmap failed");
27. close(fd);
28. printf("\nMap address = %6.6X\n",addr);
```

Продолжение

```
1.  /*Открыть разделяемую память для SEMAPHORE
2.  семафоры доступны другим – не родственным – процессам только в
3.  разделяемой памяти. Нужно открыть память для разделяемого семафора*/
4.  fd = shm_open(SEMmemory, O_RDWR | O_CREAT | O_EXCL, 0777);
5.  if(fd<0)
6.  {  printf("\ncan't open shmem for SEMAPHORE (%s). Try to close it\n",
    strerror(errno));
7.        shm_unlink(SEMmemory);
8.        fd = shm_open(SEMmemory,O_RDWR|O_CREAT|O_EXCL,0777);
9.        if(fd<0) puts("can't open shmem for SEMAPHORE");
10. }
11. /* Устанавливаем размер области */
12. if(ltrunc(fd,sizeof(SEMAPHORE),SEEK_SET)== -1)
13.       serror("set shm size error for SEMAPHORE");
14. SEM = mmap(0,sizeof(sem_t), PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
15. if(SEM == (void *)-1) serror("mmap for SEMAPHORE failed");
16. close(fd);
17. printf("\nMap address for SEMAPHORE = %6.6X\n",SEM);

18. /* Открываем семафоры */
19. if(sem_init(&SEM->access,1,1)<0) serror("sem_init/access");
20. if(sem_init(&SEM->answer,1,0)<0) serror("sem_init/answer");
21. if(sem_init(&SEM->query,1,0)<0) serror("sem_init/query");
22. printf("\nSERVER SEMAPHORES\n");
23. printf("\taccess: semid=(%d) val=(%d)\n",SEM->access.semid,SEM-> access.value);
24. printf("\tanswer: semid=(%d) value=(%d)\n",SEM->answer.semid,SEM->answer.value);
25. printf("\tquery: semid=(%d) val=(%d)\n",SEM->query.semid,SEM-> query.value);
```

Окончание

```
1. /* Основной цикл программы-сервера */
2. counter=0;
3. addr->cmd = 1;
4. while(1)
5. {    if(sem_wait(&SEM->query)<0) serror("sem_wait/query");
6.      if(addr->cmd==0) break;
7.      nval = (addr->data)+100;
8.      (addr->data) = nval;
9.      counter++;
10.     if(sem_post(&SEM->answer)<0) serror("sem_post/answer");
11.     if(sem_post(&SEM->access)<0) serror("sem_post/access");
12.}
13./* Удаляем семафоры */
14.if(sem_destroy(&SEM->access)<0) serror("sem_destroy/access");
15.if(sem_destroy(&SEM->answer)<0) serror("sem_destroy/answer");
16.if(sem_destroy(&SEM->query)<0) serror("sem_destroy/query");
17.shm_unlink(SEMmemory);
18.shm_unlink(SHMname);
19.printf("\n\nSERVER TERMINATED\nTotal queries = %d\n",counter);
20.}
```

CLIENT

```
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 <semaphore.h>
8.  #include "sc.h"
9.  SEMAPHORE       *SEM;
10. SHMdata         *addr;
11. void main(int argc, char *argv[])
12. {     int fd;
13.       int counter, nval;
14. /* Открываем разделяемую память */
15. fd = shm_open(SHMname, O_RDWR, 0777);
16. if(fd<0) serror("can't open shmem");
17. addr = mmap(0,PAGESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,0);
18. if(addr == (void *)-1) serror("mmap failed");
19. close(fd);
20. printf("\nclient[%d]: Map address = %6.6X\n",getpid(),addr);

21. /* Открыть разделяемую память для SEMAPHORE */
22. fd = shm_open(SEMmemory, O_RDWR, 0777);
23. if(fd<0) serror("can't open shmem for SEMAPHORE");
24. SEM=mmap(0,sizeof(SEMAPHORE),PROT_READ|PROT_WRITE, MAP_SHARED,fd,0);
25. if(SEM == (void *)-1) serror("mmap failed");
26. close(fd);
27. printf("\nCLIENT [%d]: Map addr for SEMAPHORE = %6.6X\n", getpid(),SEM);
```

Окончание

```
1. /* Основной цикл программы-клиента */
2. for(counter=1;counter<argc;counter++)
3. {    sleep(1); // задержка (для чистоты эксперимента)
4.      /* ждем доступа */
5.      if(sem_wait(&SEM->access)<0) serror("sem_wait/access");
6.      nval = atoi(argv[counter]);
7.      printf("\n\t-client [%d]: count me %d",getpid(),nval);
8.      addr->data = nval;
9.      addr->cmd = nval;
10.     sem_post(&SEM->query);
11.
12.     /* ждем ответ */
13.     if(sem_wait(&SEM->answer)<0) serror("sem_wait/access");
14.     printf("\n\t-client [%d]: got answ: %d",
15.                     getpid(),addr-> data);
16.}


17.printf("\n**    client [%d]: terminated\n",getpid());
18.}
```

Пример запуска сервера и клиентов

```
server &
client   3  5  6  &
client   8  3  9  &
client   5  1  0  &
```