ТЕМА 7. СИНХРОНИЗАЦИЯ НА ПРОЦЕСИ ПОСРЕДСТВОМ СЕМАФОРИ

Необходимост от синхронизация на процеси и нишки, използващи обща памет. Семафори в UNIX. Разлика при операциите над UNIX-семафори и класическите операции. Създаване на масив от семафори или достъп дп вече съществуващ. Системно извикване semget(). Изпълнение на операции над семафори. Системно извикване semop(). Изтриване на набор от семафори от системата с помоща на командата ipcrm или системното извикване semctl().

1. Необходимост от синхронизация на процеси и нишки, използващи поделена памет

В програмите разгледани в предходното упражнение са възможни ситуации, когато съвместната работа на процесите може да доведе до неверни и неочаквани резултати. Това е свързано с факта, че всяка неатомарна операция, която променя съдържанието на поделената памет, представлява обособена критична секция в процеса.

> *Съвет:* Спомнете си разгледаното в лекционния материал понятие за критична секция.

При едновременно съществуване на два процеса в операционната система може да възникне следната последователност от операции (програми в листинг 6.1a и листинг 6.1b от упражнение 6):

```
...
Процесс 1: array[0] += 1;
Процесс 2: array[1] += 1;
Процесс 1: array[2] += 1;
Процесс 1: printf(
   "Program 1 was spawn %d times,
   program 2 - %d times, total - %d times\n",
   array[0], array[1], array[2]);
...
```

Тогава програмата би извела неправилен резултат. Реалното възпроизвеждане на показаната последователност от действия в изчислителна система е много трудно, тъй като е невъзможно точно да се подбере момента от време за стартиране на процесите и да се предвиди степента на натовареност на системата. Но е възможно тази ситуация да се моделира като в двете програми пред оператора *array[2] += 1;* се добавят достатъчно дълги празни цикли. Това е илюстрирано в следващите

програми.

```
/* Программа 1 (7_1a.c), илюстрираща некоректната работа
   с поделена памет */
/* В програмата се създава поделена памет, която представлява
масив от три цели числа. В първия елемент на масива се записват
броя стартирания на програма 1, във вория елемент – броя стартирания
на програма 2, а в третия – общия брой и за двтете програми*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
int main()
{
    int *array; /* Указател към поделената памет */
    int shmid; /* IPC дескриптор за областта от поделена памет */
    int new = 1; /* Флаг, показващ необходимостта от инициализация
                    на елементите от масива */
    char pathname[] = "7_1.c"; /* /* Името на файла,
        използван за генериране на ключа. Файл с това
        име трябва да съществува в текущата директория */
    key_t key; /* IPC ключ */
    long i;
    /* Генериране на IPC ключа (използва се името на файла 7_1a.c в
            текущата директория и номера на инстанцията на областта от
        поделена памет 0 */
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Опит да се създаде поделена памет за генерирания ключ,
    ако за този ключ, тя вече съществува, системното извикване
    връща отрицателна стойност. Размера на паметта се определя
    от размера на масива, съдържащ три целочислени променливи,
    правата за достъп са 0666 – право за чтене и запис имат
    всички потребители*/
    if((shmid = shmget(key, 3*sizeof(int),
        0666|IPC_CREAT|IPC_EXCL)) < 0){
    /* В случай на грешка се определя дали, възникването й се дължи
       на това, че сегмента вече съществува или по друга причина */
        if(errno != EEXIST){
            /* Ако е по друга причина – работата се прекратява */
            printf("Can\'t create shared memory\n");
            exit(-1);
        } else {
            /* Ако е поради това, че поделената памет вече съществува,
               то се прави опит да се получи нейният IPC дескриптор.
               При успех се изчиства флага за нобходимостта от
               инициализация на елементите от масива*/
            if((shmid = shmget(key, 3*sizeof(int), 0)) < 0){
                printf("Can\'t find shared memory\n");
                exit(-1);
            }
            new = 0;
        }
    }
    /* Поделената памет се помества в адресното пространство на текущия
       процес. Обърнете внимание на това, че за да бъде сравнението
       правилно стойността -1 се преобразува в указател към цяло число.*/
    if((array = (int *)shmat(shmid, NULL, 0)) ==
        (int *)(-1)){
        printf("Can't attach shared memory\n");
        exit(-1);
    }
/* В зависимост от стойността на флага new се извършва или
инициализиране на масива, или инкрементиране на съответните броячи */
    if(new){
        array[0] = 1;
        array[1] = 0;
        array[2] = 1;
    } else {
```

```
            array[0] += 1;
            for(i=0; i<1000000000L; i++);
            /* Пределната стойност за i може да се промени в зависимост
            от производителността на компютъра */
            array[2] += 1;
        }
        /* Отпечатват се новите стойности на броячите, поделената памет се
           изтрива от адресното пространство на текущия процес и порграмата
           приключва */
        printf("Program 1 was spawn %d times,
            program 2 - %d times, total - %d times\n",
            array[0], array[1], array[2]);
        if(shmdt(array) < 0){
            printf("Can't detach shared memory\n");
            exit(-1);
        }
        return 0;
}
```

Листинг 7.1а. Програма 1 (7_1a.c), илюстрираща некоректнта работа с поделена памет

```
/* Программа 2 (7_1b.c), илюстрираща некоректната работа
   с поделена памет */
/* В програмата се създава поделена памет, която представлява масив от
три цели числа. В първия елемент на масива се записват броя стартирания
на програма 1, във вория елемент – броя стартирания на програма 2,
а в третия – общия брой и за двтете програми */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
int main()
{
int *array; /* Указател към поделената памет */
    int shmid; /* IPC дескриптор за областта от поделена памет */
    int new = 1; /* Флаг, показващ необходимостта от инициализация
                    на елементите от масива */
    char pathname[] = "7_1a.c"; /* Името на файла,
        използван за генериране на ключа. Файл с това
        име трябва да съществува в текущата директория */
    key_t key; /* IPC ключ */
    long i;
    /* Генериране на IPC ключа (използва се името на файла 05_1a.c в
            текущата директория и номера на инстанцията на областта от
        поделена памет 0 */
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Опит да се създаде поделена памет за генерирания ключ,
    ако за този ключ, тя вече съществува, системното извикване
    връща отрицателна стойност. Размера на паметта се определя
    от размера на масива, съдържащ три целочислени променливи,
    правата за достъп са 0666 – право за чтене и запис имат
    всички потребители */
    if((shmid = shmget(key, 3*sizeof(int),
        0666|IPC_CREAT|IPC_EXCL)) < 0){
    /* В случай на грешка се определя дали възникването й се дължи
       на това, че сегмента вече съществува или по друга причина */
        if(errno != EEXIST){
            /* Ако е по друга причина – работата се прекратява */
            printf("Can\'t create shared memory\n");
            exit(-1);
        } else {
            /* Ако е поради това, че поделената памет вече съществува,
               то се прави опит да се получи нейният IPC дескриптор.
               При успех се изчиства флага за нобходимостта от
               инициализация на елементите от масива */
            if((shmid = shmget(key,
```

```
                        3*sizeof(int), 0)) < 0){
                        printf("Can\'t find shared memory\n");
                        exit(-1);
                    }
                    new = 0;
                }
            }
            /* Поделената памет се помества в адресното пространство на текущия
               процес. Обърнете внимание на това, че за да бъде сравнението
               правилно стойността -1 се преобразува в указател към цяло число.*/
            if((array = (int *)shmat(shmid, NULL, 0)) ==
                (int *)(-1)){
                printf("Can't attach shared memory\n");
                exit(-1);
            }
            /* В зависимост от стойността на флага new се извършва или
            инициализиране на масива, или инкрементиране на съответните броячи */
            if(new){
                array[0] = 0;
                array[1] = 1;
                array[2] = 1;
            } else {
                array[1] += 1;
                for(i=0; i<1000000000L; i++);
                /* Пределната стойност за i може да се промени в зависимост
                от производителността на компютъра */
                array[2] += 1;
            }
            /* Отпечатват се новите стойности на броячите, поделената памет се
               изтрива от адресното пространство на текущия процес и порграмата
               приключва */
            printf("Program 1 was spawn %d times,
                program 2 - %d times, total - %d times\n",
                array[0], array[1], array[2]);
                if(shmdt(array) < 0){
                    printf("Can't detach shared memory\n");
                    exit(-1);
                }
                return 0;
        }
```

Листинг 7.1b. Програма 2 (7_1b.c), илюстрираща некоректнта работа с поделена памет

Задача 1: Наберете програмите, съхранете ги съответно с имена *7_1a.c* и *7_1b.c*, ги компилирайте. Стартирайте един път едната от тях за създаване и инициализация на споделената памет. Стартирайте другата и докато, тя се намира в цикъл стартирайте в друг виртуален терминал отново първата програма. Анализирайте резултата.

Вижда се, че за създаване на коректно работеща програма е необходимо да се обезпечи взаимоизключването на процесите при работа им с поделената памет, а така също и взаимното им изчакване за достъп до нея. Това може да се извърши с помощта на различни алгоритми за синхронизация, разгледани в лекционния материал. Например, алгоритъма на Петерсон или алгоритъма на Декер.

Задача 2 (повишена сложност): модифицирайте програмите от листинг 7.1a и листинг 7.1b, като приложите алгоритъма на Петерсон.

2. Семафори за взаимоизключване и синхронизация на процеси в UNIX

Илюстрираното в предходната точка показа необходимостта от механизами за синхронизация на работата на процеси при тяхното взаимодействие, посредством поделена памет. Един от първите механизми, предложен за синхронизиране действията на процеси е механизма на семафорите, концепцията за които е описана от Дийкстра (Dijkstra) през 1965 година. Семафорите са неотменна част от състава на System V IPC средствата. Класическия набор операции {P, V}, предложени от Дийкстра се отличава от използвания набор в System V IPC, който включва три операции:

- A(S, n) – увеличава стойността на семафора S с n;
- D(S, n) – докато S < n, процеса се блокира и S = S - n;
- Z(S) – докато S ≠ 0 процесът се блокира.

Всички IPC-семафори се инициализират с нулева стойност.

Видно е, че класическата операция P(S) съответства на операцията D(S,1), а класическата операция V(S) съответства на операцията A(S,1). Като аналог на ненулевата инициализация при семафорите на Дийкстра може да послужи създаването на семафор S и операцията A(S,n). В този случай, атомарността на двете операции трябва да се обезпечи, посредством друг семафор. Т. е., класическите семафори могат да се реализират, посредством System V IPC семафори. Обратното, обаче не е вярно, тъй като използвайки операциите P(S) и V(S) не би могло да се реализира операцията Z(S).

IPC-семафорите са съставна част от средствата в System V IPC. За тях важи всичко, което бе споменато за тези средства в материалите от предходните упражнения. IPC-семафорите са средства за връзка с непряка адресация. Те трябва да се инициализират преди да започне взаимодействието между процеси и да се предприемат необходимите действия за освобождаване на заетите от тях системни ресурси след неговия край. Пространството на имената на IPC-семафорите се състои от множеството от стойностите на ключа, генерирани с помощта на функцията *ftok()*. Операциите над семафори се извършват посредством системни извиквания, които получават като параметър IPC- дескриптора

на семафора, който го идентифицира еднозначно в цялата изчислителна система, а цялата информация относно семафорите се разполага в адресното пространство на ядрото на операционната система. Това позволява да се организира взаимодействие между процеси, които не се намират едновременно в системата.

3. Създаване и достъп до масив от семафори

В операционната система UNIX се създава не по един семафор за всяка конкретна стойност на ключа, а цял масив (в Linux – до 500 семафора, като това количество може да бъде намалено от системния администратор). Това се прави с цел икономия на ресурси. За създаване на масив от семафори, асоцииран с определен ключ или достъп до вече съществуващ масив, според зададения ключ се използва системното извикване *semget()* (аналог на системното извикване *shmget()* при използване на поделена памет), което връща стойността на IPC-дескриптора за този масив. При него се използват същите способи за създаване и достъп, които се използват и при работата с поделена памет. Новосъздадените семафори се инициализират с нулева стойност.

ИМЕ

semget

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

ОПИСАНИЕ

Системното изивкване semget е предназанчено за достъп до масив от IPC-семафори. В случай на успешно приключване, връща System V IPC дескриптора за този масив (цяло неотрицателно число, еднозначно характеризиращо масива от семафори вътре в изчислителната система).
Параметърът key е System V IPC ключа за масива от семафори т. е. неговото име от пространството на System V IPC имената. Неговата стойност се получава или с помоща на функцията ftok() или се задава специалната константа IPC_PRIVATE. Използването на IPC_PRIVATE води до създаване на нов сегмент поделена памет с ключ, който не съвпада с ключа на нито един от вече съществуващите сегменти и който не може да бъде получен с помоща на функцията ftok() при нито една комбинация на нейните параметри.
Параметърът nsems определя броя на семафорите в създавания или вече съществуващия масив. В случай, че масива с укзания ключ вече съществува, но неговия размер не съвпада с указания в параметъра nsems, възниква грешка.
Параметърът semflg се използва само при създаване на нов масив от семафори. Той определя правата на различните потребители за достъп до масива. Освен това, чрез него се определя дали масива съществува и съответно дали има нужда от създаване на нов. Стойността на този параметър се задава като комбинация от (с помощта на побитово или – "|") следните предварително дефинирани константи и осмичните права за достъп:
IPC_CREAT – ако масив с указания ключ не съществува, трябва да бъде създаден;

IPC_EXCL – използва се съвместно с флага IPC_CREAT. При съвместното им използване и наличието на масива с указания ключ, възниква грешка, при което променливата errno, описана във файла <errno.h>, приема стойност EEXIST;
0400 – право за четене за потребителя, създал масива;
0200 – право за запис за потребителя, създал масива;
0040 – право за четене за групата на потребителя, създал масива;
0020 – право за запис за групата на потребителя, създал масива;
0004 – право за четене за всички останали потребители;
0002 – право за запис за всички останали потребители.
Новосъздаден семафор се инициализира със стойност нула.

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойнстта на System V IPC дескриптора за масива от семафори, при нормално завършване и стойност -1 при възникване на грешка.

4. Операции над семафори

За изпълнение на операциите A, D и Z над семафори от даден масив се използва системното извикване *semop()*, което има сравнително сложна семантика. Разработчиците на System V IPC са натоварили това извикване не само с изпълнение на споменатите три операции, но и с допълнителни такива и то едновременно върху няколко семафора в масива. За правилното използване на това извикване е необходимо:

1. Да се определи кои семафори от масива предстои да се използват и какви операции следва да се извършат над тях. Необходимо е да се вземе предвид, че операциите над семафори реално се изпълняват само след успешно приключване на системното изивкване. Например, при изпълнение на операциите A(S1,5) и Z(S2) в едно извикване, ако S2 != 0, то стойността на семафора S1 няма да се измени, докато S2 стане равно на 0. Редът на изпълнение на операциите, в случая, когато процеса не прминава в състояние на очакване (не се блокира) е неопределен. Така, например, при едновремено изпълнение на операциите A(S1,1) и D(S2,1), ако S2 > 1, не е известно кое би протекло по рано – намаляването на стойността на семафора S2 или увеличаването на стойността на семафора S1. Ако трябва да се съблюдава реда за протичане на операциите, най-добре е да се използват няколко извиквания вместо едно.

2. Да се дефинира масив от тип *struct sembuf* с размер равен на определения брой семафори. Операциите, извършвани над всеки елемент от масива, съответстват на операциите над един семафор.

3. Да се запълнят елементите на масива. В полето *sem_flg* на всеки елемент е необходимо да се зададе стойност 0. В полетата *sem_num* и *sem_op* следва да се зададе номера на семафора в масива от IPC

семафори и съответния код на операцията. Номерирането на семафорите започва от 0, т.е ако в масива има само един семафор, той ще бъде с номер 0. Операциите над семафори се кодират по следния начин:

• за изпълнение на операцията A(S,n) стойността на полето *sem_op* трябва да бъде равна на n;

• за изпълнение на операцията D(S,n) стойността на полето *sem_op* трябва да бъде равна на –n;

• за изпълнение на операцията Z(S) стойността на полето *sem_op* трябва да бъде равна на 0.

4. Да се зададе адреса на запълнения масив, като втори параметър на системното извикване *semop()*, а като трети – определения предварително брой на използвани семафори.

ИМЕ

semop

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, int nsops);
```

ОПИСАНИЕ

Системното извикване semop е предназначено за изпълнение на операциите A, D и Z. Даденото описание не е пълно и е адаптирано за настоящия курс. Повече информация, включително примери за неговото използване, може да се намери в UNIX Manual.
Параметърът semid е System V IPC дескриптора за набора от семафори, т. е. стойността, която връща системното извикване semget() при създаване на набора от семафори или при достъпа до него по зададен ключ.
Всеки от елементите на масива (nsops на брой) към който сочи указателя sops, определя операцията, която трябва да бъде извършена над който и да е семафор от IPC масива. В структурата struct sembuf на указания от sops масив влизат следните променливи:

* short sem_num – номер на семафора в масива от IPC семафори;
* short sem_op – изпълняваната операция;
* short sem_flg – за този флаг в настоящото упражнение винаги се използва стойност 0.

Стойността на sem_op се определя по следния начин:

* за операцията A(S,n), е необходимо sem_op = n;
* за операцията D(S,n), е необходимо sem_op = -n;
* за операцията Z(S), е необходимо sem_op = 0.

Операциите над семафори се изпълняват само при успешно приключване на системното извикване. Ако един процес премине в състояние на очакване след изпълнение на операциите D или Z и възникне една от следните ситуации:

* масива от семафори бъде изтрит от системата;
* процеса получи сигнал, който трябва да бъде обработен.

Процеса излиза от състоянието на очакване, а системното извикване приключва с възникване на грешка.

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойнст 0, при нормално завършване и стойност -1 при възникване на грешка.

За илюстрация са предстваени две примерни програми,

синхронизиращи своите действия, посредством семафори.

```
/* Программа 7_2a.c, илюстрираща работата със семафори */
/* Тази програма получава достъп до един системен семафор,чака,
докато се стартира програмата 7_2b.c и неговата стойност стане
по-голяма или равна на 1, след което установява семафора в 1*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
int main()
{
       int semid; /* IPC дескриптор за масива от IPC семафори */
       char pathname[] = "7_2a.c"; /* Името на файла,
        използван за генериране на ключа. Файл с това
        име трябва да съществува в текущата директория */
       key_t key; /* IPC ключ */
       struct sembuf mybuf; /* Дефиниране на структурата за
              операциите над семафори */
       /* Генериране на IPC ключа (използва се името на файла
          7_2a.c в текущата директория) и номера на инстанцията
          за масива от семафори 0 */
       if((key = ftok(pathname,0)) < 0){
              printf("Can\'t generate key\n");
              exit(-1);
       }
       /* Опит да се получи достъп до масива от семафори по зададения
       ключ, или ако такъв масив не съществува да бъде създаден, с
       права за достъп read & write за всички потребители */
       if((semid = semget(key, 1, 0666 | IPC_CREAT)) < 0){
              printf("Can\'t get semid\n");
              exit(-1);
       }
       /* Изпълнение на операцията D(semid1,1) за използвания масив
       от семафори. За тази цел се попълва структурата от данни. За
       флага се задава стойност 0. Масива се състои от един семафор
       с номер 0. Кода на операцията е -1.*/
       mybuf.sem_op = -1;
       mybuf.sem_flg = 0;
       mybuf.sem_num = 0;
       if(semop(semid, &mybuf, 1) < 0){
              printf("Can\'t wait for condition\n");
              exit(-1);
       }
       printf("Condition is present\n");
       return 0;
}
```

Листинг 7.2a. Програма 1 (7_2a.c), илюстраща работата със семафори

```
/* Програма 7_2b.c, илюстрираща работата със семафори */
/* Тази програма получава достъп до един системен семафор
   и го увеличава на 1*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
int main()
{
       int semid; /* IPC дескриптор за масива от IPC семафори */
       char pathname[] = "7_2a.c"; /* Името на файла,
        използван за генериране на ключа. Файл с това
        име трябва да съществува в текущата директория */
       key_t key; /* IPC ключ */
       struct sembuf mybuf; /* Дефиниране на структурата за
              операциите над семафори */
       /* Генериране на IPC ключа (използва се името на файла
          7_2a.c в текущата директория) и номера на инстанцията
          за масива от семафори 0 */
       if((key = ftok(pathname,0)) < 0){
              printf("Can\'t generate key\n");
```

```
                exit(-1);
        }
        /* Опит да се получи достъп до масива от семафори по зададения
        ключ, или ако такъв масив не съществува да бъде създаден, с
        права за достъп read & write за всички потребители */
        if((semid = semget(key, 1, 0666 | IPC_CREAT)) < 0){
                printf("Can\'t get semid\n");
                exit(-1);
        }
        /* Изпълнение на операцията A(semid1,1) за използвания масив
        от семафори. За тази цел се попълва структурата от данни. За
        флага се задава стойност 0. Масива се състои от един семафор
        с номер 0. Кода на операцията е 1.*/
        mybuf.sem_op = 1;
        mybuf.sem_flg = 0;
        mybuf.sem_num = 0;
        if(semop(semid, &mybuf, 1) < 0){
                printf("Can\'t wait for condition\n");
                exit(-1);
        }
        printf("Condition is set\n");
        return 0;
}
```

Листинг 7.2b. Програма 2 (7_2b.c), илюстраща работата със семафори

Първата програма изпълнява над семафора S операцията D(S,1), а втората – изпълнява над същия семафор операцията A(S,1). Ако семафора не съществува в системата, една от двете програми го създава преди да изпълни операцията. Тъй като при създаването на семафор, той се инициализира с 0, програма 1 се деблокира едва след стартиране на програма 2.

Задача 3: Наберете програмите, съхранете ги съответно под имена 7_2a.c и 7_2b.c, компилирайте и проветете поведението им. Изменете програмите така, че първата програма да се деблокира след 5 старта на втората.

5. Изтриване на семафорите от системата

Масивът от семафори, както бе илюстрирано с предходните примери, може да продължи да съществува в системата и след завършване на използвщите го процеси, като семафорите продължават да съхраняват стойностите си. Това може да доведе до некоректно поведение на програмите, ако се предполага, че семафорите туко що са били създадени и следователно имат нулеви стойности. Затова, преди стартиране на подобна програма е необходимо семафорите в системата да се изтрият. За изтриване на семафори могат да се използват семафорите *ipcs* и *ipcrm*, разгледани в материала на едно от предходните упражнения. Командата *ipcrm* в този случай трябва да има вида:

```
ipcrm sem <IPC идентификатор>
```

За същата цел може да се използва и системното извикване *semctl()*, което може да изпълнява и други операции над масив от семафори, но тяхното разглеждане излиза извън рамките на настоящото ръководство.

ИМЕ

semctl

ПРОТОТИП

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, union semun arg);
```

ОПИСАНИЕ

Системното извикване shmctl е предназначено за получаване на информация за масив от IPC семафори, промяна на неговите атрибути и изтриването му от системата. Даденото описание не е пълно, а адаптирано за целите на настоящия курс. Пълно описание може да се намери в UNIX Manual.
В това упражнение системното изивикване shmctl се използва само за изтриване на масив от семафори от системата. Параметърът semid е System V IPC дескриптора за масива от семафори, т. е. стойността, която връща системното изивкване shmget() при създаване на масива или при достъп до него по зададения ключ.
В рамките на това упражнение за параметъра cmd винаги ще се задава стойност IPC_RMID – командата за изтриване на масива от семафори заделен със своя идентфикатор. Параметърът semnum за тази команда не е необходим и винаги приема стойност 0.
Ако съществуват процеси, които се намират в състояние на очакване заради семафори от изтривания масив, то те ще бъдат деблокоирани и системното извикване semop(), ще приключи с възникване на грешка.

ВРЪЩАНИ СТОЙНОСТИ

Системното извикване връща стойност 0 при нормално приключване и стойност -1 при възникване на грешка.

В точка 1 от настоящото упражнение бе показано, че всяка неатомарна операция, която модифицира съдържнието на споделена памет, представлва обособена критична секция в процеса.

Задача 4: Промененте програмата от точка 1 "Необходимост от синхронизация на процеси и нишки, използващи обща памет", като реализирате взаимоизключване с помоща на семафори, така че, тя да работи коректно с поделената памет.

Неименуваните канали са еднопосочни канали за връзка (упражнение 4). Организирането на двупосочна връзка, посредством един канал изисква механизъм за взаимна синхронизация на процесите.

Задача 5: Организирайте двупосочна връзка с взаимно изчакване на процеса-родител и процеса-дете чрез pipe, като за взаимното изчакване (синхронизацията) използвате семафори. За целата модифицирайте програмата от листинг 4.3 от упражнение 4.