

# Операційні системи. Практикум

## 5. Семафори в UNIX як засіб синхронізації процесів

### План заняття

Семафори в UNIX. Відмінність операцій над UNIX-семафорами від класичних операцій.....	1
Створення масиву семафорів або доступ до уже існуючого. Системний виклик <code>semget()</code> .....	2
Виконання операцій над семафорами. Системний виклик <code>semop()</code> .....	3
Приклад з використанням семафора .....	5
Зміна попереднього прикладу .....	6
Видалення набору семафорів із системи за допомогою команди <code>ipcrm</code> або системного виклику <code>semctl()</code> .....	6
Написання, компіляція і виконання програми з організацією взаємовиключення за допомогою семафорів для двох процесів, що взаємодіють через розділювану пам'ять .....	7
Написання, компіляція і виконання програми з організацією взаємної черговості за допомогою семафорів для двох процесів, що взаємодіють через <code>pipe</code> .....	8
Поняття про POSIX-семафори .....	8

### Семафори в UNIX. Відмінність операцій над UNIX-семафорами від класичних операцій

У матеріалах попереднього семінару мова йшла про необхідність синхронізації роботи процесів для їхньої коректної взаємодії через розділювану пам'ять. Як згадувалося в лекції 6, одним з перших механізмів, запропонованих для синхронізації поведінки процесів, стали семафори, концепцію яких описав Дейкстра (Dijkstra) в 1965 році. При розробці засобів System V IPC семафори ввійшли в їхню будову як невід'ємна частина. Слід зазначити, що набір операцій над семафорами System V IPC відрізняється від класичного набору операцій  $\{P, V\}$ , запропонованого Дейкстрою. Він включає три операції:

- $A(S, n)$  -- збільшити значення семафора  $S$  на величину  $n$ ;
- $D(S, n)$  -- поки значення семафора  $S < n$ , процес блокується. Далі  $S = S - n$ ;
- $Z(S)$  -- процес блокується доти, поки значення семафора  $S$  не стане рівним 0.

Всі IPC-семафори ініціалізуються нульовим значенням.

Ми бачимо, що класичній операції  $P(S)$  відповідає операція  $D(S, 1)$ , а класичній операції  $V(S)$  відповідає операція  $A(S, 1)$ . Аналогом ненульової ініціалізації семафорів Дейкстри значенням  $n$  може служити виконання операції  $A(S, n)$  відразу після створення семафора  $S$ , із забезпеченням атомарності створення семафора і її виконання за допомогою іншого семафора. Ми показали, що класичні семафори реалізуються через семафори System V IPC. Зворотна дія не є вірною. Використовуючи операції  $P(S)$  і  $V(S)$ , ми не зуміємо реалізувати операцію  $Z(S)$ .

Оскільки IPC-семафори є складовою частиною засобів System V IPC, то для них вірно все, що говорилося про ці засоби в матеріалах попереднього семінару. IPC-семафори є засобом зв'язку з непрямою адресацією, вимагають ініціалізації для організації взаємодії процесів і спеціальних дій для звільнення системних ресурсів по його закінченні. Простором імен IPC-семафорів є безліч значень ключа, які генеруються за допомогою

функції `ftok()`. Для здійснення операцій над семафорами системним викликом як параметр передаються IPC- дескриптори семафорів, які однозначно ідентифікують їх у всій обчислювальній системі, а вся інформація про семафори розташовується в адресному просторі ядра операційної системи. Це дозволяє організовувати через семафори взаємодію процесів, які навіть не перебувають у системі одночасно.

## Створення масиву семафорів або доступ до уже існуючого. Системний виклик `semget()`

З метою економії системних ресурсів операційна система UNIX дозволяє створювати не по одному семафору для кожного конкретного значення ключа, а зв'язувати із ключем цілий масив семафорів (в Linux -- до 500 семафорів у масиві, хоча ця кількість може бути зменшена системним адміністратором). Для створення масиву семафорів, асоційованого з певним ключем, або доступу за ключем до вже існуючого масиву використовується системний виклик `semget()`, що є аналогом системного виклику `shmget()` для розділюваної пам'яті, який повертає значення IPC-дескриптора для цього масиву. При цьому застосовуються ті ж способи створення і доступу (див. семінари 6-7 розділ "Розділювана пам'ять в UNIX. Системні виклики `shmget()`, `shmat()`, `shmdt()`"), що і для розділюваної пам'яті. Знову створені семафори ініціюються нульовим значенням.

### Системний виклик `semget()`

#### Прототип системного виклику

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

#### Опис системного виклику

Системний виклик `semget` призначений для виконання операції доступу до масиву IPC-семафорів і, у випадку її успішного завершення, повертає дескриптор System V IPC для цього масиву (ціле невід'ємне число, що однозначно характеризує масив семафорів всередині обчислювальної системи і використовується надалі для інших операцій з ним).

Параметр `key` є ключем System V IPC для масиву семафорів, тобто фактично його іменем із простору імен System V IPC. Як значення цього параметра може використовуватися значення ключа, отримане за допомогою функції `ftok()`, або спеціальне значення `IPC_PRIVATE`. Використання значення `IPC_PRIVATE` **завжди** приводить до спроби створення нового масиву семафорів із ключем, що не збігається зі значенням ключа жодного із уже існуючих масивів і не може бути отриманий за допомогою функції `ftok()` ні при одній комбінації її параметрів.

Парааметр `nsems` визначає кількість семафорів у створюваному або вже існуючому масиві. У випадку, якщо масив із зазначеним ключем уже є, але його розмір не збігається із зазначеним у параметрі `nsems`, констатується виникнення помилки.

Параметр `semflg` -- прапорці -- відіграє роль тільки при створенні нового масиву семафорів і визначає права різних користувачів при доступі до масиву, а також необхідність створення нового масиву і поведінку системного виклику при спробі створення. Він є деякою комбінацією (за допомогою операції побітове або - "|") наступних визначених значень і вісімкових прав доступу:

`IPC_CREAT` -- якщо масиву для зазначеного ключа не існує, він повинен бути створений

`IPC_EXCL` -- застосовується разом із прапорцем `IPC_CREAT`. При спільному їхньому використанні і існуванні масиву із зазначеним ключем, доступу до масиву не відбувається

і констатується помилка, при цьому змінна `errno`, описана у файлі `<errno.h>`, прийме значення `EEXIST`

- 0400 -- дозволене читання для користувача, що створив масив
  - 0200 -- дозволений запис для користувача, що створив масив
  - 0040 -- дозволене читання для групи користувача, що створив масив
  - 0020 -- дозволений запис для групи користувача, що створив масив
  - 0004 -- дозволене читання для всіх інших користувачів
  - 0002 -- дозволений запис для всіх інших користувачів
- Заново створені семафори ініціалізуються нульовим значенням.

Значення, що повертається

Системний виклик повертає значення дескриптора System V IPC для масиву семафорів при нормальному завершенні і значення -1 при виникненні помилки.

## Виконання операцій над семафорами. Системний виклик `semop()`

Для виконання операцій A, D і Z над семафорами з масиву використовується системний виклик `semop()`, який володіє досить складною семантикою. Розробники System V IPC явно переважили цей виклик, застосовуючи його не тільки для виконання всіх трьох операцій, але ще і для декількох семафорів у масиві IPC-семафорів одночасно. Для правильного використання цього виклику необхідно виконати наступні дії:

1. Визначитися, для яких семафорів з масиву мають бути виконані операції. Необхідно мати на увазі, що всі операції реально відбуваються тільки перед успішним поверненням із системного виклику, тобто якщо ви хочете виконати операції  $A(S1, 5)$  і  $Z(S2)$  в одному виклику і виявилось, що  $S2 \neq 0$ , то значення семафора  $S1$  не буде змінено доти, поки значення  $S2$  не стане рівним 0. Порядок виконання операцій у випадку, коли процес не переходить у стан **очікування**, не визначено. Так, наприклад, при одночасному виконанні операцій  $A(S1, 1)$  і  $D(S2, 1)$  у випадку  $S2 > 1$  невідомо, що відбудеться раніше -- зменшиться значення семафора  $S2$  або збільшиться значення семафора  $S1$ . Якщо порядок для вас важливий, краще застосувати кілька викликів замість одного.
2. Після того як ви визначилися з кількістю семафорів і виконуваних операцій, необхідно завести в програмі масив з елементів типу `struct sembuf` з розмірністю, що дорівнює певній кількості семафорів (якщо операція відбувається тільки над одним семафором, можна, природно, обійтися просто змінною). Кожний елемент цього масиву буде відповідати операції над одним семафором.
3. Заповнити елементи масиву. У поле `sem_flg` кожного елемента потрібно занести значення 0 (інші значення прапорців у семінарах ми розглядати не будемо). У поля `sem_num` і `sem_op` варто занести номери семафорів у масиві IPC семафорів і відповідні коди операцій. Семафори нумеруються, починаючи з 0. Якщо у вас у масиві всього один семафор, то він буде мати номер 0. Операції кодуються так:
  - для виконання операції  $A(S, n)$  значення поля `sem_op` повинно дорівнювати  $n$ ;
  - для виконання операції  $D(S, n)$  значення поля `sem_op` повинне бути дорівнює  $-n$ ;
  - для виконання операції  $Z(S)$  значення поля `sem_op` повинне бути дорівнює 0.
4. Як другий параметр системного виклику `semop()` вказати адресу заповненого масиву, а як третій параметр -- раніше визначену кількість семафорів, над якими здійснюються операції.

## Системний виклик semop()

### Прототип системного виклику

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

### Опис системного виклику

Системний виклик semop призначений для виконання операцій A, D і Z (див. опис операцій над семафорами з масиву IPC семафорів -- розділ "Створення масиву семафорів або доступ до уже існуючого. Системний виклик semget()" цього семінару). Даний опис не є повним описом системного виклику, а обмежується рамками поточного курсу. Для повного опису звертайтеся до UNIX Manual.

Параметр semid є дескриптором System V IPC для набору семафорів, тобто значенням, яке повернув системний виклик semget() при створенні набору семафорів або при його пошуку за ключем.

Кожний з nsops елементів масиву, на який вказує параметр sops, визначає операцію, яка повинна бути виконана над яким-небудь семафором з масиву IPC семафорів, і має тип структури struct sembuf, у яку входять наступні змінні:

short sem\_num -- номер семафора в масиві IPC семафорів (нумеруються, починаючи з 0);

short sem\_op -- виконувана операція;

short sem\_flg -- прапорці для виконання операції. У нашому курсі завжди будемо вважати цю змінну рівною 0.

Значення елемента структури sem\_op визначається в такий спосіб:

для виконання операції A(S, n) значення повинно дорівнювати n;

для виконання операції D(S, n) значення повинно дорівнювати -n;

для виконання операції Z(S) значення повинно дорівнювати 0.

Семантика системного виклику має на увазі, що всі операції будуть насправді виконані над семафорами тільки перед успішним поверненням із системного виклику. Якщо при виконанні операцій D або Z процес перейшов у стан очікування, то він може бути виведеним із цього стану при виникненні наступних форс-мажорних ситуацій:

масив семафорів був вилучений із системи;

процес одержав сигнал, який повинен бути оброблений.

У цьому випадку відбувається повернення із системного виклику з констатацією помилкової ситуації.

### Значення, що повертається

Системний виклик повертає значення 0 при нормальному завершенні і значення -1 при виникненні помилки.

## Приклад з використанням семафора

Для ілюстрації сказаного розглянемо найпростіші програми, що синхронізують свої дії за допомогою семафорів

```
/* Програма 05-1a.c для ілюстрації роботи з
семафорами */
/* Ця програма одержує доступ до одного системного семафора,
чекає, поки його значення не стане більшим або рівним 1
після запусків програми 05-1b.c, а потім зменшує його на 1*/
#include <sys/types.h>
```

```

#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
int main()
{
    int semid; /* IPC дескриптор для масиву IPC семафорів */
    char pathname[] = "08-1a.c"; /* Ім'я файлу,
        що використовується для генерації ключа. Файл із таким
        іменем повинен існувати в поточній директорії */
    key_t key; /* IPC ключ */
    struct sembuf mybuf; /* Структура для задання
        операції над семафором */
    /* Генеруємо IPC-ключ із імені файлу 05-1a.c у поточній
        директорії і номера екземпляра масиву семафорів 0 */
    if((key = ftok(pathname,0)) < 0){
        printf("Can\'t generate key\n");
        exit(-1);
    }
    /* Намагаємося одержати доступ за ключем до масиву
        семафорів, якщо він існує, або створити його з одного
        семафора, якщо його ще не існує, із правами доступу
        read & write для всіх користувачів */
    if((semid = semget(key, 1, 0666 | IPC_CREAT)) < 0){
        printf("Can\'t get semid\n");
        exit(-1);
    }
    /* Виконаємо операцію D(semid,1) для нашого масиву
        семафорів. Для цього спочатку заповнимо нашу структуру.
        Прапорець встановлюємо рівним 0. Наш масив семафорів
        складається з одного семафора з номером 0. Код операції -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;
}

```

*Лістинг 5.1. Програма 05-1a.c для ілюстрації роботи із семафорами*

```

/* Програма 05-1b.c для ілюстрації роботи з семафорами */
/* Ця програма одержує доступ до одного системного семафора
і збільшує його на 1*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
int main()
{
    int semid; /* IPC дескриптор для масиву IPC
        семафорів */
    char pathname[] = "08-1a.c"; /* Ім'я файлу,
        що використовується для генерації ключа. Файл із таким
        іменем повинен існувати в поточній директорії */
    key_t key; /* IPC ключ */
    struct sembuf mybuf; /* Структура для задання операції

```

```

над семафором */
/* Генеруємо IPC ключ із імені файлу 05-1a.c у поточній
директорії і номера екземпляра масиву семафорів 0 */
if((key = ftok(pathname,0)) < 0){
    printf("Can\'t generate key\n");
    exit(-1);
}
/* Намагаємося одержати доступ за ключем до масиву
семафорів, якщо він існує, або створити його з
одного семафора, якщо його ще не існує, із правами доступу
read & write для всіх користувачів */
if((semid = semget(key, 1, 0666 | IPC_CREAT)) < 0){
    printf("Can\'t get semid\n");
    exit(-1);
}
/* Виконаємо операцію A(semid,1) для нашого масиву
семафорів. Для цього спочатку заповнимо нашу структуру.
Прапорець встановлюємо рівним 0. Наш масив
семафорів складається з одного семафора з номером 0.
Код операції 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 5.1b. Програма 05-1b.c для ілюстрації роботи із семафорами*

Перша програма виконує над семафором S операцію D(S,1), друга програма виконує над тим же семафором операцію A(S,1). Якщо семафора в системі не існує, будь-яка програма створює його перед виконанням операції. Оскільки при створенні семафор завжди ініціалізується 0, то програма 1 може працювати без блокування тільки після запуску програми 2. Наберіть програми, збережіть їх під іменами 05-1a.c і 05-1b.c відповідно, відкомпілюйте і перевірте правильність їхнього поведінки.

### **Зміна попереднього прикладу**

Змініть програми з попереднього розділу так, щоб перша програма могла працювати без блокування після не менш ніж 5 запусків другої програми.

### **Видалення набору семафорів із системи за допомогою команди `ipcrm` або системного виклику `semctl()`**

Як ми бачили в прикладах, масив семафорів може продовжувати існувати в системі і після завершення процесів, які його використовували, а семафори будуть зберігати своє значення. Це може привести до некоректного поведінки програм, які припускають, що семафори були тільки що створені і тому мають нульове значення. Необхідно видаляти семафори із системи перед запуском таких програм або перед їх завершенням. Для видалення семафорів можна скористатися командами `ipcs` і `ipcrm`, розглянутими в матеріалах попереднього семінару. Команда `ipcrm` у цьому випадку повинна мати вигляд `ipcrm sem <IPC ідентифікатор>`

Для цієї ж мети ми можемо застосовувати системний виклик `semctl()`, що вміє виконувати й інші операції над масивом семафорів, але їхній розгляд виходить за рамки нашого курсу.

Системний виклик `semctl()`

Прототип системного виклику

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

Опис системного виклику

Системний виклик `semctl` призначений для одержання інформації про масив IPC семафорів, зміни його атрибутів і видалення його із системи. Даний опис не є повним описом системного виклику, а обмежується рамками поточного курсу. Для вивчення повного опису звертайтеся до UNIX Manual.

У нашому курсі ми будемо застосовувати системний виклик `semctl` тільки для видалення масиву семафорів із системи. Параметр `semid` є дескриптором System V IPC для масиву семафорів, тобто значенням, яке повернув системний виклик `semget()` при створенні масиву або при його пошуку за ключем.

Як параметр `cmd` у рамках нашого курсу ми завжди будемо передавати значення `IPC_RMID` -- команду для видалення сегмента розділюваної пам'яті із заданим ідентифікатором. Параметри `semnum` і `arg` для цієї команди не використовуються, тому ми завжди будемо підставляти замість них значення 0.

Якщо які-небудь процеси перебували в стані очікування для семафорів з масиву, який видаляється, при виконанні системного виклику `semop()`, то вони будуть разблоковані і повернуться з виклику `semop()` з індикацією помилки.

Значення, що повертається

Системний виклик повертає значення 0 при нормальному завершенні і значення -1 при виникненні помилки.

### **Написання, компіляція і виконання програми з організацією взаємовиключення за допомогою семафорів для двох процесів, що взаємодіють через розділювану пам'ять**

У матеріалах семінарів 6-7 було показано, що будь-які неатомарні операції, пов'язані зі зміною вмісту розділюваної пам'яті, є критичною секцією процесу або нитки виконання. Модифікуйте програми з розділу "Необхідність синхронізації процесів і ниток виконання, що використовують загальну пам'ять" семінарів 6-7, які ілюстрували некоректну роботу через розділювану пам'ять, забезпечивши за допомогою семафорів взаємовиключення для їхньої правильної роботи.

### **Написання, компіляція і виконання програми з організацією взаємної черговості за допомогою семафорів для двох процесів, що взаємодіють через ріре**

У матеріалах семінару 5, коли мова йшла про зв'язок родинних процесів через ріре, відзначалося, що ріре є однонаправленим каналом зв'язку, і що для організації зв'язку через один ріре у двох напрямках необхідно використовувати механізми взаємної синхронізації процесів. Організуйте двосторонній почерговий зв'язок процесу-батька і

процесу-дитини через `pipe`, використовуючи для синхронізації семафори, модифікувавши програму з розділу "Програма для організації однонапрявленого зв'язку між рідними процесами через `pipe`" семінару 5.

## **Поняття про POSIX-семафори**

У стандарті POSIX вводяться інші семафори, повністю аналогічні семафорам Дейкстри. Для ініціалізації значення таких семафорів застосовується функція `sem_init()`, аналогом операції P служить функція `sem_wait()`, а аналогом операції V -- функція `sem_post()`. На жаль, в Linux такі семафори реалізовані тільки для ниток виконання одного процесу, і тому докладно ми на них зупинятися не будемо.