Семафоры в Linux

Семафор – это механизм, который позволяет параллельным процессам и потокам работать с общими ресурсами и решать вопросы синхронизации.

Виды семафоров:

- Бинарные семафоры (двоичные)
- Семафоры-счетчики (числовые)
- Массивы семафоров (множественные)

Двоичные семафоры

Двоичный семафор (sv) — это переменная, способная принимать только значения 0 и 1.

Операции с двоичным семафором P(sv)-закрыть и V(sv)-открыть.

P(sv): Если sv больше нуля, она уменьшается на единицу. Если sv равна 0, выполнение данного процесса приостанавливается.

V(sv): Если какой-то другой процесс был приостановлен в ожидании семафора sv, то он активизируется. Если ни один процесс не приостановлен в ожидании семафора sv, значение переменной увеличивается на единицу.

Если несколько процессов одновременно запрашивают **P** или **V** операции над одним и тем же семафором, то эти операции будут выполняться **последовательно в произвольном порядке.**

Числовые семафоры

Числовой семафор (sc) — это переменная, способная принимать значения 0 - N.

Операции с числовым семафором P(sc, n), V(sc, k) и Z(sc,0) – дождаться закрытия.

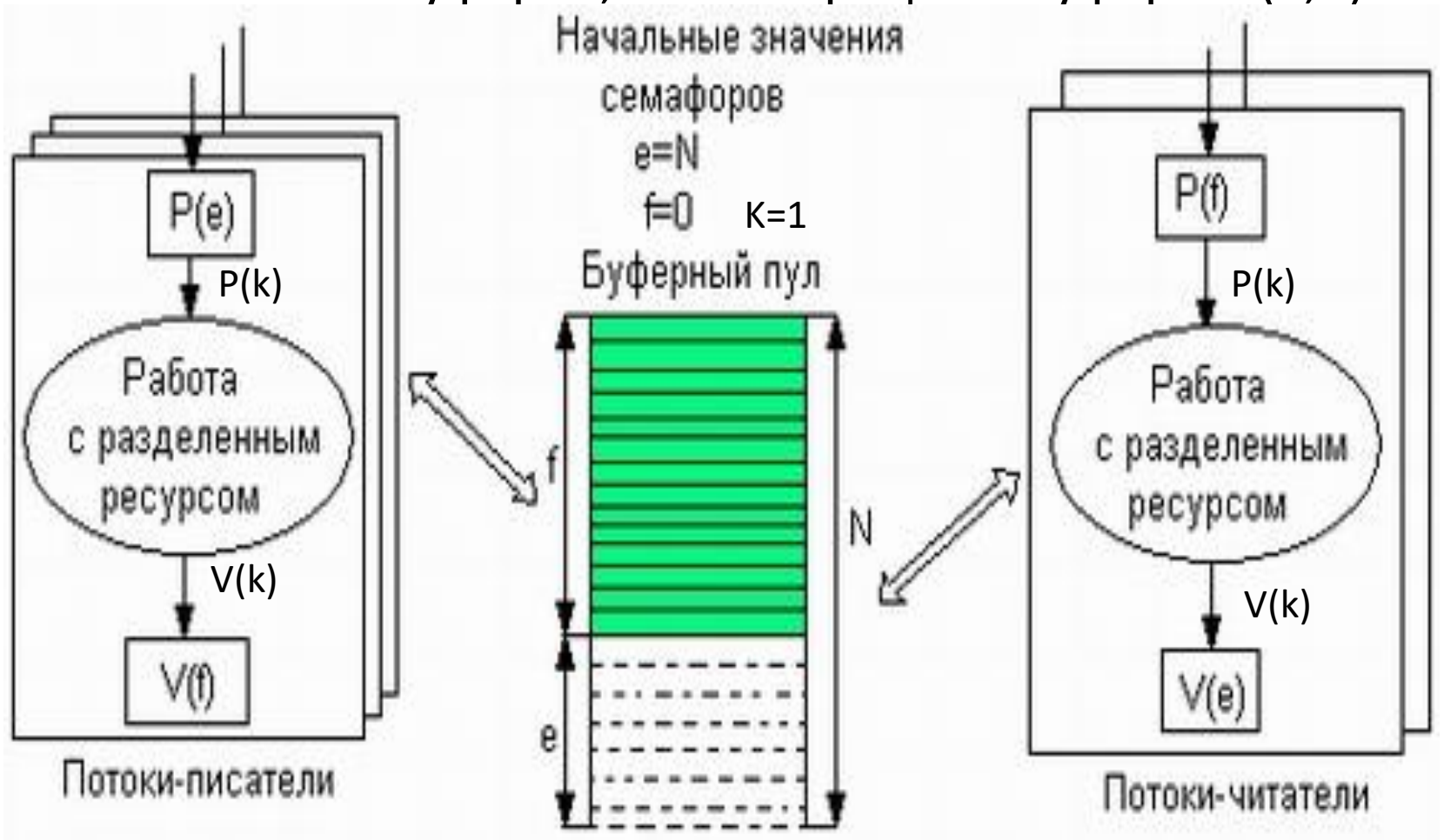
P(sc, n): Если sc- n≥0, то sc=sc- n, в противном случае процесс приостанавливается.

V(sc, k): – sc=sc+k и активизируются процессы ожидающие открытие семафора, они пытаются выполнить P(sc, n).

Z(sc,0): проверка семафора на 0. Если значение не равно нулю, то процесс переходит в состояние ожидания.

Использование числовых семафоров для синхронизации процессов

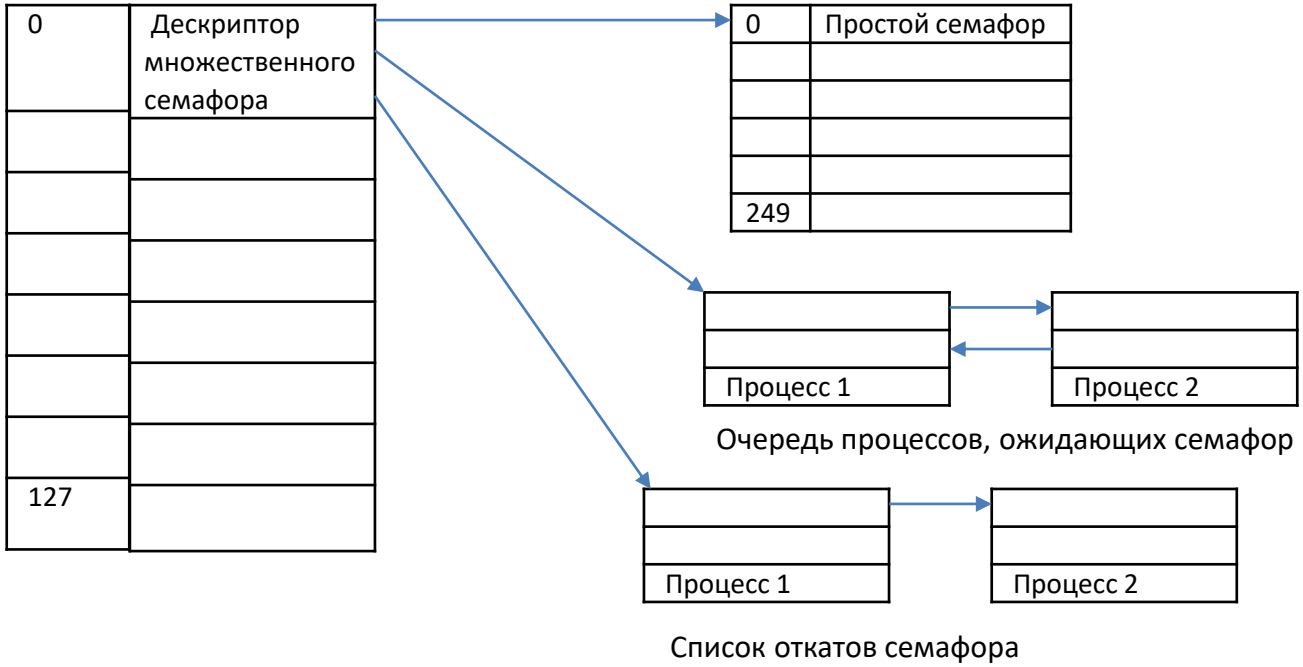
три семафора: e — число пустых буферов, f — число заполненных буферов, к — операция с буфером (0,1)



Множественные семафоры

Множественный семафор - это объект специального типа, который одновременно доступен нескольким процессам и состоит из одного или нескольких простых семафоров.

Процесс, выполняющий операцию с множественным семафором, либо продолжается, если **все команды** с его простыми семафорами закончились успешно, либо переводится в состояние ожидания, если хотя бы одна команда не выполнилась. Ожидающий процесс будет находиться в очереди к множественному семафору, пока не появится возможность выполнить все команды с простыми семафорами.



Дескриптор множественного семафора

/* По одной структуре данных для каждого набора семафоров в системе. */

struct sem_array {

struct kern_ipc_perm sem_perm; /* права доступа */

time_t sem_otime; /* время последнего обращения */

time_t sem_ctime; /* время последнего изменения */

struct sem *sem_base; /* указатель на первый семафор в массиве */

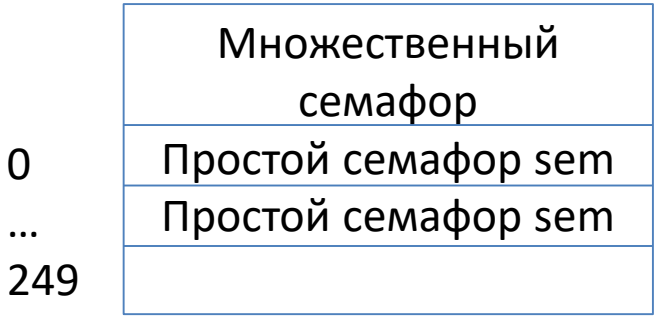
struct sem_queue *sem_pending; /* операции, ожидающие исполнения */

struct sem_queue **sem_pending_last; /* последняя ожидающая операция */

struct sem_undo *undo; /* список отката для данного массива * /
unsigned long sem_nsems; /* кол-во семафоров в массиве */

};

Представление множественного семафора




```
struct sem {
int semval; /* текущее значение от 0 до 32767 */
int sempid; /* pid процесса последней операции */
};
```

Очередь процессов ожидающих семафор

```
/* представляет собой узел в очереди ожидающих операций. */
struct sem_queue {
struct sem_queue * next; /* следующий элемент очереди */
struct sem_queue ** prev; /* предыдующий элемент очереди */
struct task_struct* sleeper; /* этот процесс */
struct sem_undo * undo; /* структура откатов */
int pid; /* pid процесса */
int status; /* результат выполнения операции */
struct sem_array * sma; /* массив семафоров для выполнения операций */
int id; /* внутренний sem id */
struct sembuf * sops; /* массив ожидающих операций */
int nsops; /* кол-во операций */
int alter; /* признак изменения семафора */
};
```

Структура элементов отката

Если процесс установил флаг SEM_UNDO, то при завершении этого процесса ядро даст обратный ход всем операциям, выполненным процессом. Это позволяет избежать блокирования семафора процессом, который закончил свою работу прежде, чем освободил захваченный им семафор.

Структура, хранящая изменения семафора, имеет вид:

```
struct sem_undo {
struct sem_undo * proc_next; /* следующий элемент списка для данного процесса */
struct sem_undo * id_next; /* следующий элемент в данном наборе семафоров */
int semid; /* ID набора семафоров */
short * semadj; /* массив изменений, по одному на семафор */
};
```

Предельные характеристики множественных семафоров

ipcs -s –l

------ Пределы семафоров --------

максимальное количество = 128 массивов

максимум семафоров на массив = 250

максимум семафоров на всю систему = 32000

максимум операций на вызов семафора = 32

максимальное значение семафора = 32767

Создание множества семафоров

int semget (key_t key, int nsems, int semflg);

Возвращает: идентификатор множественного семафора в случае успеха и -1 в случае ошибки errno:

EACCESS (доступ отклонен)

EEXIST (существует нельзя создать)

EIDRM (множество помечено как удаляемое)

ENOENT (множество не существует, не было исполнено ни одного IPC_CREAT)

ENOMEM (не хватает памяти для новых семафоров)

ENOSPC (превышен лимит на количество множеств семафоров)

Параметры операции создания множества семафоров

int semget (key_t key, int nsems, int semflg);

key – ключ > 0

nsems – число семафоров в множестве < 250

semflg – права доступа и флаги IPC_CREAT и IPC_EXCL

Права доступа к множественному семафору

0400 - владельцу разрешено читать характеристики семафора Z(sc,0);

0200 - владельцу разрешено изменять характеристики семафора P(sc, n), V(sc, k) ;

0040 - члену группы владельца разрешено читать характеристики семафора Z(sc,0);

0020 - члену группы владельца разрешено изменять характеристики семафора P(sc, n), V(sc, k) ;

0004 - всем остальным пользователям разрешено читать характеристики семафора Z(sc,0);

0002 - всем остальным пользователям разрешено изменять характеристики семафора P(sc, n), V(sc, k)

```
int idsem= semget(100, 2, IPC_CREAT|0666);
```

Выполнение операций с множественным семафором

int semop(int semid, struct sembuf *sops, unsigned nsops);

Возвращает: 0 в случае успеха (все операции выполнены) -1 в случае ошибки

errno:

E2BIG (nsops больше чем максимальное число позволенных операций)

EACCESS (доступ отклонен)

EAGAIN (при флаге IPC_NOWAIT операция не может быть выполнена)

EFAULT (sops указывает на ошибочный адрес)

EIDRM (множество семафоров уничтожено)

EINTR (сигнал получен во время в состоянии ожидания)

EINVAL (множество не существует или неверный semid)

ENOMEM (поднят флаг SEM_UNDO, но не хватает памяти для создания необходимой undo-структуры)

ERANGE (значение семафора вышло за пределы допустимых значений)

Параметры операции с множественным семафором

int semop(int semid, struct sembuf *sops, unsigned nsops);

semid – идентификатор семафора;

sops – массив операций;

nsops – число операций.

struct sembuf {

unsigned short sem_num; /* индекс семафора в массиве */

short sem_op; /* операция (положительное, отрицательное число или нуль)*/

short sem_flg; /* флаги */

};

Выполняемая операция определяется следующим образом:

- Положительное значение поля sem_op предписывает увеличить значение семафора на величину sem_op.
- Отрицательное значение поля sem_op предписывает уменьшить значение семафора на абсолютную величину sem_op. Операция не может быть успешно выполнена, если в результате получится отрицательное число.
- Нулевое значение поля sem_op предписывает сравнить значение семафора с нулем. Операция не может быть успешно выполнена, если значение семафора отлично от нуля.

Флаги: IPC_NOWAIT и SEM_UNDO

struct sembuf op1={0,0,0}, /* проверка на 0 первого семафора***/**
op2={0,2,0}, /* открытие первого семафора***/**
op3={1,-1, IPC_NOWAIT};/*условное закрытие второго семафора***/**

semop(id_sem, &op1, 1);

Семантика выполнения операций с семафорами

1. Если все команды из списка завершились успешно, то функция возвратит 0.
2. Если одна из команд списка завершилась неуспешно и для нее был установлен флаг **IPC_NOWAIT**, то выполнение операции будет прервано на этой команде и функция возвратит значение -1, причем ни у одного из семафоров не будет изменено значение.
3. Если одна из команд списка завершилась неуспешно и для нее не был задан флаг **IPC_NOWAIT**, то операция прерывается, процесс переходит в состояние ожидания и становится в конец очереди к семафору, состоящую из элементов типа **sem_queue**. В состоянии ожидания процесс будет находиться до тех пор, пока не появится возможность успешно завершить все команды из списка операции. Проверку этой возможности запускает другой процесс, успешно завершивший операцию с данным множественным семафором.

Контроль и управление семафорами

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

Возвращает: натуральное число в случае успеха и -1 в случае ошибки

errno =

EACCESS (доступ отклонен)

EFAULT (адрес, указанный аргументом arg, ошибочен)

EIDRM (множество семафоров удалено)

EINVAL (множество не существует или неправильный semid)

EPERM (EUID не имеет привилегий для cmd в arg-е)

ERANGE (значение семафора вышло за пределы допустимых значений)

Чтобы выполнить управляющее действие IPC_SET или IPC_RMID, процесс должен иметь действующий идентификатор пользователя, равный либо идентификаторам создателя или владельца очереди, либо идентификатору суперпользователя. Для выполнения управляющих действий SETVAL и SETALL требуется право на изменение, а для выполнения остальных действий - право на чтение.

Параметры операции контроля и управления семафорами

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

semid – идентификатор семафора;

semnum – номер семафора;

cmd – команда (IPC_STAT, IPC_SET, IPC_RMID, GETALL, GETPID, GETVAL, SETALL, SETVAL)

arg – информация о семафорах

union semun {

int val; /* значение для GETVAL ,SETVAL */

struct semid_ds *buf; /* буфер для IPC_STAT и IPC_SET */

ushort *array; /* массив для GETALL и SETALL */

struct seminfo *__buf; /* буфер для IPC_INFO */

void *__pad;

};

Установка и чтение значений семафора

При создании множественного семафора все его простые семафоры имеют значение 0.

Установка 1 для простого семафора 0

```
semctl ( semid, 0, SETVAL, 1 )
```

Установка значений для трех простых семафоров

```
short values[3] = { 2, 0, 1 };
semctl( semid, 3, SETALL, values );
```

Чтение простого семафора 0

```
int val ;
val =semctl( semid, 0, GETVAL, 0 );
```

Уничтожение семафора

shmctl(semid,IPC_RMID,0);

Изменение характеристик и уничтожение семафора допускается только процессу, у которого эффективный идентификатор пользователя принадлежит либо root, либо создателю или владельцу сегмента.

После выполнения этой операции множественный семафор становится недоступным, память, выделенная для дескриптора, освобождается, а процессы, стоящие в очереди к семафору, активизируются. Они заканчивают выполнение операций с семафором с признаком ошибки.

Пример синхронизации процесов с помощью семафоров (процесс 1)

```
int main() {
int semid; /* идентификатор семафоров */
struct sembuf mybuf; /* Структура для задания операции над
    семафором */
/* Создаем один семафор с ключом 100 */
if((semid = semget(100, 1, 0666 | IPC_CREAT)) < 0){
 printf(“Семафор не создан\n"); exit(-1); }
mybuf.sem_op = -1; /* закрываем семафор */
mybuf.sem_flg = 0;
mybuf.sem_num = 0;
if(semop(semid, &mybuf, 1) < 0){/* ожидаем открытия */
printf(“Операция не выполнилась\n"); exit(-1); }
printf(“Завершение работы программы 1\n");
return 0;
}
```

Пример синхронизации процесов с помощью семафоров (процесс 2)

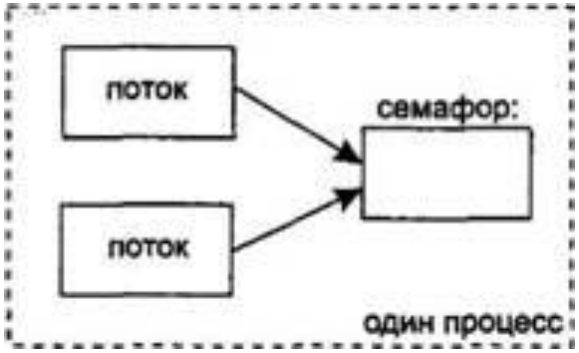
```
int main() {
int semid; /* идентификатор семафоров */
struct sembuf mybuf; /* Структура для задания операции над семафором */
/* Получаем идентификатор семафора с ключом 100 */
if((semid = semget(100, 1, 0666 | IPC_CREAT)) < 0){
 printf(“Семафор не создан\n"); exit(-1); }
mybuf.sem_op = 1; /* увеличиваем семафор */
mybuf.sem_flg = 0;
mybuf.sem_num = 0;
if(semop(semid, &mybuf, 1) < 0){/* открываем семафор */
printf(“Операция не выполнилась\n"); exit(-1); }
printf(“Завершение работы программы 2\n");
return 0;
}
```

Функции для работы с семафорами POSIX

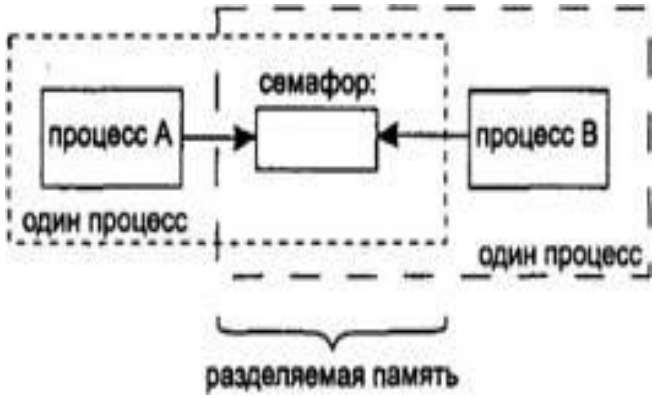
Метод	Описание
sem_open	Открытие/создание именованного семафора.
sem_close	Закрытие именованного семафора.
sem_unlink	Удаление именованного семафора.
sem_init	Инициализация неименованного семафора.
sem_destroy	Удаление неименованного семафора.
sem_getvalue	Получение текущего значения счётчика семафора.
sem_wait	Выполнение операции блокировки семафора (P).
sem_trywait	Выполнение операции попытки блокировки семафора(P) без перехода в ожидание.
sem_post	Освобождение семафора (V).

Функции sem_init и sem_destroy

int sem_init(sem_t **sem,* int *shared,* unsigned int *value*);



Shared=0
**sem располагается в приложении*



Shared=1;
sem располагается в разделяемой памяти

int sem_destroy(sem_t **sem*);

Семафор взаимного исключения mutex

Мьютекс представляет собой двоичный семафор для синхронизации потоков.
#include <pthread.h>
Есть три типа мьютекса:
/* Стандартный мьютекс */
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
/* Мьютекс со счетчиком*/
pthread_mutex_t lock = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
/* Мьютекс проверки ошибки */
pthread_mutex_t lock = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;

Они отличаются только когда владелец мьютекса повторно вызывает операцию захвата pthread_mutex_lock:

- Для стандартного мьютекса возникает тупиковая ситуация, поскольку поток теперь ждёт самого себя, чтобы разблокировать мьютекс.
- Для мьютекса со счетчиком функция возвращается сразу и счётчик захвата мьютекса увеличивается на единицу. Мьютекс разблокируется только если счетчик дойдёт до нуля; то есть поток должен вызвать pthread_mutex_unlock для каждого вызова pthread_mutex_lock.
- Для мьютекса проверки ошибки pthread_mutex_lock возвращает ошибку с кодом ошибки EDEADLK.

Операции с mutex

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
Функция инициализирует мьютекс аттрибутом **mutexattr**.
Если mutexattr равен *NULL*, то мьютекс инициализируется значением по умолчанию.
В случае успешного выполнения функции (код возрата 0),
мьютекс считается инициализированным и «свободным».

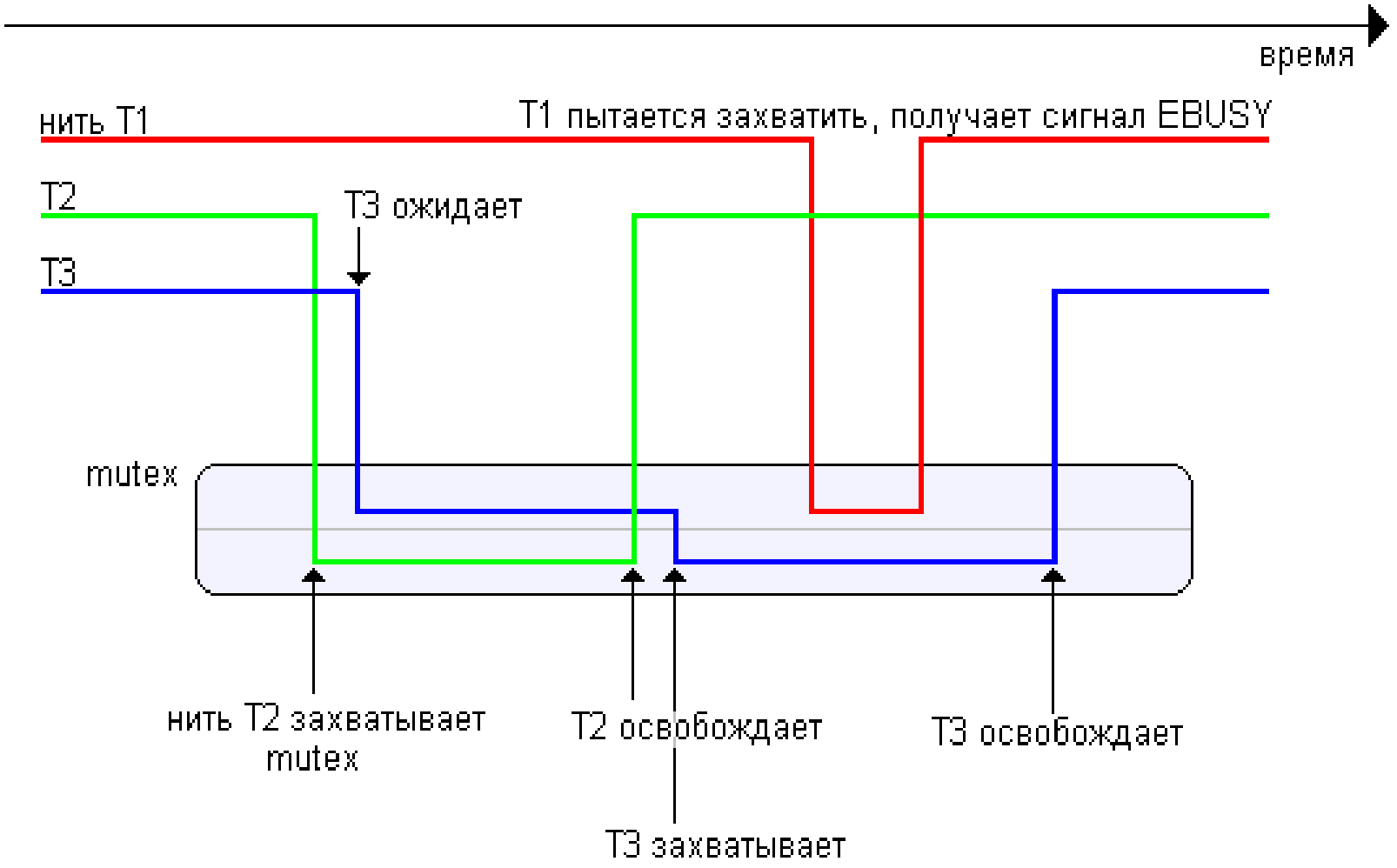
int pthread_mutex_lock(pthread_mutex_t *mutex);
Если mutex не занят, то процесс занимает его, становится его обладателем и продолжает работу.
Если мьютекс занят, то процесс блокируется и ждет освобождения мьютекса.

int pthread_mutex_trylock(pthread_mutex_t *mutex);
Идентична по поведению функции **pthread_mutex_lock()**,
с одним исключением — она не блокирует процесс, если **mutex** занят,
а возвращает *EBUSY* код.

int pthread_mutex_unlock(pthread_mutex_t *mutex);
Освобождает занятый мьютекс.

int pthread_mutex_destroy(pthread_mutex_t *mutex);
Уничтожение мьютекса.

Пример работы с мьютексом



Задание 1 к лабораторной работе 10

Напишите две программы (Поставщик и Потребитель), которые работают с циклическим буфером ограниченного размера, расположенным в разделяемой памяти. Поставщик выделяет буфер и семафоры, читает по одному символу из файла и записывает его в буфер. Потребитель считывает по одному символу из буфера и выводит их на экран. Если буфер пустой, то Потребитель должен пассивно ждать, пока Поставщик не занесет туда хотя бы один символ. Если буфер полностью заполнен, то пассивно должен ждать Поставщик, пока Потребитель не извлечет из него по крайней мере один символ. Поставщик заканчивает свою работу, как только прочитает последний символ из файла и убедится, что Потребитель его прочитал. Потребитель заканчивает свою работу при отсутствии символов в буфере и завершении работы Поставщика. Синхронизация процессов должна выполняться с помощью семафоров.

Задание 2 к лабораторной работе 10

Напишите две программы, экземпляры которых запускаются параллельно и с разной частотой обращаются к общему файлу. Каждый процесс из первой группы (Писатель) пополняет файл определенной строкой символов и выводит ее на экран вместе с именем программы. Процессы второй группы (Читатели) считывают строки из файла и выводят их на экран при условии отсутствия ожидающих запись Писателей. Пока один Писатель записывает строку в файл, другим Писателям и всем Читателям запрещено обращение к файлу. Если Писатели не пишут в файл, то разрешается одновременная работа всех Читателей. **Писатели должны ожидать, пока не закончат работу запущенные Читатели**. Писатель заканчивает работу после того как выполнит N-кратную запись строки в файл. Работа Читателя завершается, когда он прочитал весь текущий файл. Синхронизация процессов должна выполняться с помощью семафоров.