

Лабораторная 11

Используемые функции :

Сигнатура функции :

```
#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

функция `pthread_mutexattr_destroy()` должна уничтожить объект атрибутов мьютекса; объект становится в результате неинициализированный. Реализация может привести к тому, что `pthread_mutexattr_destroy()` установит ссылку на объект присваивается атрибуту недопустимое значение.

Уничтоженный объект атрибутов `attr` можно повторно инициализировать с помощью `pthread_mutexattr_init()`; результат в противном случае ссылки на объект после его уничтожения не определены.

Функция `pthread_mutexattr_init()` должна инициализировать `attr` объекта атрибутов мьютекса значением по умолчанию. (значение для всех атрибутов, определенных реализацией)

Результаты не определены, если `pthread_mutexattr_init()` вызывается с указанием уже инициализированного атрибута.

После того, как объект атрибутов мьютекса был использован для инициализации одного или нескольких мьютексов, любая функция, влияющая на объект атрибутов (включая уничтожение) не должен влиять ни на какие ранее инициализированные мьютексы.

Поведение не определено, если значение, указанное аргументом `attr` функции `pthread_mutexattr_destroy()`, не ссылаться на инициализированный объект атрибута мьютекса.

Возвращаемое значение :

0 - успех

код ошибки - ошибка

Возможные ошибки :

Функция pthread_mutexattr_init() завершится ошибкой, если:

ENOMEM - Недостаточно памяти для инициализации объекта атрибутов мьютекса.

Эти функции не должны возвращать код ошибки [EINTR].

Сигнатура функции :

```
#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
                             int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive (see below)	error returned
DEFAULT	non-robust	undefined behavior†	undefined behavior†
DEFAULT	robust	undefined behavior†	error returned

PTHREAD_MUTEX_NORMAL

PTHREAD_MUTEX_ERRORCHECK

Позволяет избежать взаимоблокировок, возвращая ненулевое значение, если один и тот же поток пытается заблокировать один и тот же мьютекс более одного раза без предварительной разблокировки мьютекса. Или же если один и тот же поток пытается разблокировать один и тот же мьютекс более одного раза без предварительной блокировки.

PTHREAD_MUTEX_RECURSIVE

Позволяет одному и тому же pthread рекурсивно блокировать мьютекс с помощью подпрограммы pthread_mutex_lock, не приводя к взаимоблокировке или получению ненулевого возвращаемого значения из pthread_mutex_lock. То есть можем несколько раз одним потоком делать лок на один и тот же мьютекс, только этот же поток должен вызвать pthread_mutex_unlock столько же раз, сколько он вызывал pthread_mutex_lock, чтобы разблокировать мьютекс для использования другими потоками.

PTHREAD_MUTEX_DEFAULT

Если тип мьютекса равен PTHREAD_MUTEX_DEFAULT, поведение pthread_mutex_lock() может соответствовать одному из трех других стандартные типы мьютексов, как описано в таблице выше. Если он не соответствует одному из этих трех, поведение не определено для отмеченных случаев. **Это значение по умолчанию**

Возвращаемое значение :

После успешного завершения pthread_mutexattr_gettype() функция должна возвращать ноль и сохранять значение переданного типа атрибутов в переданную структуру атрибутов. В противном случае будет возвращен код ошибки.

функция pthread_mutexattr_settype() :

0 - успех

код ошибки - ошибка

Возможные ошибки :

Функция pthread_mutexattr_settype() завершится ошибкой, если:

EINVAL устанавливаемый тип атрибутов мьютекса недопустим.

Эти функции не должны возвращать код ошибки [EINTR].

Сигнатура функции :

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Функция pthread_mutex_init() должна инициализировать мьютекс на который ссылается мьютекс с атрибутами, указанными attr. Если attr == NULL, используются атрибуты мьютекса по умолчанию;

После успешной инициализации состояние мьютекса становится инициализированным и разблокированным

В тех случаях, когда атрибуты мьютекса по умолчанию являются подходящими, для инициализации мьютексов можно использовать макрос PTHREAD_MUTEX_INITIALIZER

. Эффект должен быть эквивалентен динамической инициализации вызовом pthread_mutex_init() с параметром attr, указанным как NULL , за исключением того, что проверки ошибок (ошибок как от функции pthread_mutex_init) не выполняются.

Функция pthread_mutex_destroy() уничтожает объект мьютекса , на который ссылается мьютекс; объект мьютекса становится, по сути, неинициализированным. Реализация может привести к тому, что функция pthread_mutex_destroy() установит для объекта, на который ссылается мьютекс , недопустимое значение.

Уничтоженный объект мьютекса может быть повторно инициализирован с помощью

pthread_mutex_init()); результаты других ссылок на объект после его уничтожения не определены.

Возвращаемое значение :

0 - успех

код ошибки - ошибка

Возможные ошибки :

Функция pthread_mutex_init() завершится ошибкой, если:

EAGAIN системе не хватало необходимых ресурсов (кроме памяти) для инициализации другого мьютекса.

Для инициализации мьютекса недостаточно памяти.

EPERM Вызывающий поток не имеет привилегий выполнять эту операцию.

Функция pthread_mutex_init() может завершиться с ошибкой, если:

EINVAL Объект атрибутов, на который ссылается ttr установлен надежный (robust mutex attribute set) атрибут мьютекса без установки атрибута, совместно используемого процессом (process-shared attribute)

.

Эти функции не должны возвращать код ошибки[EINTR].

Аттрибуты :

The attributes allow you to set or get:

- the [type](#) (deadlocking, deadlock-detecting, recursive, etc).
- the [robustness](#) (what happens when you acquire a mutex and the original owner died while possessing it).
- the [process-shared attribute](#) (for sharing a mutex across process boundaries).
- the [protocol](#) (how a thread behaves in terms of priority when a higher-priority thread wants the mutex).
- the [priority ceiling](#) (the priority at which the critical section will run, a way of preventing priority inversion).

Функции `pthread_mutexattr_getprioceiling()` и `pthread_mutexattr_setprioceiling()` соответственно должны получить и установить атрибут максимального приоритета атрибутов мьютекса, на который указывает объект `attr`, который был ранее создан функцией `pthread_mutexattr_init()`.

Атрибут `prioceiling` содержит максимальный приоритет инициализированных мьютексов. Значения `priosce` находятся в пределах максимального диапазона приоритетов, определенных `SCHED_FIFO`.

По умолчанию - 1

```
SYNOPSIS
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr,
                                int *pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
                                int pshared);

Compile and link with -pthread.

DESCRIPTION
These functions get and set the process-shared attribute in a mutex attributes object. This
attribute must be appropriately set to ensure correct, efficient operation of a mutex created
using this attributes object.
```

PTHREAD_PROCESS_PRIVATE

Мьютексы, созданные с помощью этого объекта атрибутов, должны совместно использоваться только потоками в том же процессе, который инициализировал мьютекс. **Это значение по умолчанию для атрибута общего процесса мьютекса.**

PTHREAD_PROCESS_SHARED

Мьютексы, созданные с помощью этого объекта атрибутов, могут совместно использоваться любыми потоками, имеющими доступ к памяти, содержащей объект, включая потоки в разных процессах.

Сигнатура функции :

```
#include <pthread.h>

int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
                                attr, int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
                                int robust);
```

PTHREAD_MUTEX_STALLED

Это значение по умолчанию для объекта атрибутов мьютекса. Если мьютекс инициализируется с атрибутом PTHREAD_MUTEX_STALLED, а его владелец умирает, не разблокировав его, мьютекс после этого остается заблокированным, и любые будущие попытки вызвать pthread_mutex_lock(3) для мьютекса будут блокироваться на неопределенный срок.

PTHREAD_MUTEX_ROBUST

Если мьютекс инициализируется с атрибутом PTHREAD_MUTEX_ROBUST, а его владелец умирает, не разблокировав его, любые будущие попытки вызвать pthread_mutex_lock(3) для этого мьютекса будут успешными и вернут EOWNERDEAD, чтобы указать, что первоначальный владелец больше не существует и мьютекс находится в несогласованном состоянии. Обычно после возврата EOWNERDEAD следующий владелец должен вызвать pthread_mutex_consistent(3) для полученного мьютекса, чтобы снова сделать его согласованным, прежде чем использовать его дальше.

Если следующий владелец разблокирует мьютекс с помощью pthread_mutex_unlock(3) до того, как сделает его согласованным, мьютекс будет навсегда непригоден для использования, и любые последующие попытки заблокировать его с помощью pthread_mutex_lock(3) завершатся с ошибкой ENOTRECOVERABLE. Единственная разрешенная операция с таким мьютексом — это pthread_mutex_destroy(3).

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Мьютекс позволяет создать синхронизованный доступ к разделяемым ресурсам так, что после блокировки мьютекса одним потоком другие не могут заблокировать его и выполнять следующую часть кода до разблокировки этого мьютекса

Если мьютекс уже заблокирован другим потоком, вызывающий поток блокируется до тех пор, пока мьютекс не станет доступным. Эта операция должна возвращаться с мьютексом, на который ссылается переданный мьютекс в заблокированном состоянии с вызывающим потоком в качестве его владельца.

(Если поток пытается повторно заблокировать мьютекс, который он уже заблокировал, *pthread_mutex_lock*


() должен вести себя так, как описано в **Повторная блокировка** столбец следующей таблицы. Если поток пытается разблокировать мьютекс, который он не заблокировал, или мьютекс, который разблокирован, *pthread_mutex_unlock*

() должен вести себя так, как описано в **Разблокировать, если не является владельцем** столбец следующей таблицы.)

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive (see below)	error returned
DEFAULT	non-robust	undefined behavior†	undefined behavior†
DEFAULT	robust	undefined behavior†	error returned

what is the "attribute" of a pthread mutex?

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other

 <https://stackoverflow.com/questions/4252005/what-is-the-attribute-of-a-pthread-mutex>



Функции `pthread_mutex_lock()` и `pthread_mutex_trylock()` завершатся ошибкой, если:

EAGAIN Не удалось получить мьютекс, поскольку было превышено максимальное количество рекурсивных блокировок для мьютекса.

EINVAL Мьютекс был создан с атрибутом протокола, имеющим значение `PTHREAD_PRIO_PROTECT`, и приоритет вызывающего потока выше, чем текущий потолок приоритета мьютекса.

НЕ ПОДЛЕЖИТ ВОССТАНОВЛЕНИЮ

Состояние, защищенное мьютексом, восстановлению не подлежит.

EOWNERDEAD

Мьютекс является надежным мьютексом, и процесс, содержащий

предыдущий поток-владелец, завершается при удержании блокировки мьютекса. Блокировка мьютекса должна быть получена вызывающим потоком, и новый владелец должен сделать состояние согласованным.

Функция `pthread_mutex_lock()` завершится ошибкой, если:

EDEADLK

Тип мьютекса - `PTHREAD_MUTEX_ERRORCHECK`, и текущий поток уже владеет мьютексом.

Функция `pthread_mutex_trylock()` завершится ошибкой, если:

ИСПОЛЬЗУЙТЕ мьютекс не удалось получить, поскольку он уже был заблокирован.

Функция `pthread_mutex_unlock()` завершится ошибкой, если:

НАПРИМЕР, тип мьютекса - `PTHREAD_MUTEX_ERRORCHECK` или `PTHREAD_MUTEX_RECURSIVE`, или мьютекс является надежным мьютексом, и текущий поток не владеет мьютексом.

Функции `pthread_mutex_lock()` и `pthread_mutex_trylock()` могут завершиться с ошибкой, если:

EOWNERDEAD

Мьютекс является надежным мьютексом, и предыдущий поток-владелец завершается при удержании блокировки мьютекса. Блокировка мьютекса

должна быть получена вызывающим потоком, и новый владелец должен сделать состояние согласованным.

Функция `pthread_mutex_lock()` может завершиться с ошибкой, если:

EDEADLK

Было обнаружено состояние взаимоблокировки.

Эти функции не должны возвращать код ошибки `[EINTR]`.