The POSIX threads permit objects

v0.91

Generated by Doxygen 1.8.1.1

Mon Jul 9 2012 12:25:04

Contents

Chapter 1

The POSIX threads permit objects

Herein lies a suite of safe, composable, user mode permit objects for POSIX threads. They are used to asynchronously give a thread of code the *permission* to proceed and as such are very useful in any situation where one bit of code must asychronously notify another bit of code of something.

Unlike other forms of synchronisation, permit objects do not suffer from race conditions. Unlike naive use of condition variables, permit objects do not suffer from lost or spurious wakeups - there is **always** a one-to-one relationship between the granting of a permit and the receiving of that permit. Unlike naive use of semaphores, permit objects do not suffer from producer-consumer problems. While having similarities to event objects, unlike event objects permit objects always maintain the concept of *atomic ownership* of the permit - exactly *one* actor may own a permit at any one time. This strong guarantee allows permits to be much safer and more predictable in many-granter many-waiter scenarios than event objects.

There are three permit objects:

1. A simple implementation, pthread_permit1_t. This is the simplest and fastest implementation of a POSIX threads permit. It is typically compiled inline, and it is always consuming, non-hookable and non-selectable.

2. A pthread_permitc_t denotes a consuming POSIX threads permit. It is hookable and selectable. Selectable means that pthread_permitX_select() may be called upon an array of permits which returns after any one of the supplied permits receives a grant.

3. A pthread_permitnc_t denotes a non-consuming POSIX threads permit. It is also hookable and selectable. Non-consuming permits can also optionally mirror their state onto a kernel file descriptor, allowing the use of select() and poll().

1.1 Features

POSIX threads permit objects are guaranteed to not use dynamic memory except in those functions which explicitly say so. They can also optionally never sleep the calling thread, instead spinning until the permit is gained. This permits their use in bootstrap or small embedded systems, or where low latency is paramount.

This reference implementation is written in C11 and the latest version can be found at https://github.com/ned14/ISO_POSIX_standards_stuff/tree/master/pthreads%20Notifer%20Object. It contains a full set of unit tests written using C++ CATCH. The software licence is Boost's software licence (free to use by anyone).

It contains support for Microsoft Windows 7 and POSIX. It has been tested on Microsoft Visual Studio 2010, GCC v4.6 and clang v3.2.

The simple permit object costs 48/0/142 CPU cycles for grant/revoke/wait uncontended and 359/4/372 cycles when contended between two threads. These results are for an Intel Core 2 processor.

1.2 Why is it necessary that a permit object be added to POSIX threads?

There are many occasions in threaded programming when a third party library goes off and does something asynchronous in the background. In the meantime, the foreground thread may do other tasks, occasionally polling a notification object to see if the background job has completed, or indeed if it runs out of foreground things to do, it may simply sleep until the completion of the background job or jobs. Put simply, the foreground threads polls or waits for permission to continue.

The problem is that naive programmers think a wait condition is suitable for this purpose. This is highly incorrect due to the problem of spurious and lost wakeups inherent to wait conditions. Despite the documentation for pthread_cond saying this, wait conditions are frequently proposed as the "correct" solution in many "expert advice" internet sites including stackflow.com among others. The present lack of standardised, safe asynchronous notification objects in POSIX leads to too many "roll your own" implementations which are too frequently subtly broken. This leads to unreliability in threaded programming. Furthermore, there is a problem of interoperability - how can third party libraries interoperate easily when each rolls its own asynchronous notification object.

A completely safe notification object can be built from atomics and wait conditions - indeed, what is proposed is entirely built this way. The problem is rather one of standardisation on a safe, efficient, and well tested implementation.

1.3 Acknowledgements

The POSIX threads permit objects proposed by this document came from internal deliberations by WG14 during the preparation of the C11 standard - I am highly indebted to those on the committee who gave so freely of their time and thoughts. My thanks in particular are due to Hans Boehm without whose detailed feedback this proposal would look completely different. I would also like to thank Anthony Williams for his commentary and feedback, and Nick Stoughton for his advice to me regarding becoming the ISO JTC1 SC22 convenor for the Republic of Ireland and on how best to submit a proposal for incorporation into the POSIX standard. My thanks are also due to John Benito, WG14 convenor, for his seemingly never tiring efforts on the behalf of C-ish programmers everywhere.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Permit types

The types of pthread_permit.

Typedefs

- typedef struct pthread_permit1_s pthread_permit1_t

  The simplest and fastest implementation of a POSIX threads permit. Always consuming, non-hookable and non-selectable.
- typedef struct pthread_permitc_s pthread_permitc_t

  A consuming POSIX threads permit. Hookable and selectable.
- typedef struct pthread_permitnc_s pthread_permitnc_t

  A non-consuming POSIX threads permit. Hookable and selectable.
- typedef void * pthread_permitX_t

  A pointer to any of pthread_permit1_t, pthread_permitc_t and pthread_permitnc_t.
- typedef int(* pthread_permitX_grant_func)(pthread_permitX_t)

  A permit grant function prototype for any of the permit grant functions.

5.1.1 Detailed Description

The types of pthread_permit. A pthread_permit1_t denotes the simplest and fastest implementation of a POSIX threads permit. It is typically compiled inline, and it is always consuming, non-hookable and non-selectable.

A pthread_permitc_t denotes a consuming POSIX threads permit, while a pthread_permitnc_t denotes a non-consuming POSIX threads permit. Both are hookable and selectable. Non-consuming permits are additionally optionally kernel file descriptor mirroring.

pthread_permitX_t denotes any of pthread_permit1_t, pthread_permitc_t and pthread_permitnc_t. It is specifically for pthread_permit1_grant(pthread_permitX_t permit), pthread_permitc_grant(pthread_permitX_t permit) and pthread_permitnc_grant(pthread_permitX_t permit) all of which match the permit grant function prototype type definition pthread_permitX_grant_func. The API of the permit grant has been so unified to allow the following idiom:

```
int ask_3rd_party_library_to_do_an_asynchronous_job(..., pthread_permitX_t
      permit, pthread_permitX_grant_func completionroutine);
...
pthread_permitnc_t permit;
pthread_permitnc_init(&permit);
ask_3rd_party_library_to_do_an_asynchronous_job(..., &permit,
      pthread_permitnc_grant);
...
// Wait for completion
pthread_permitnc_wait(&permit);
```

In other words, client code can supply whichever type of permit it prefers to third party library code. That third party library simply calls the supplied completion routine, thus granting the permit.

5.2 Permit hooks

The hooks of pthread_permitc_t and pthread_permitnc_t.

Data Structures

- struct pthread_permitc_hook_t

  The hook data structure.
- struct pthread_permitnc_hook_t

  The hook data structure.

Typedefs

- typedef struct pthread_permitc_hook_s pthread_permitc_hook_t

  The hook data structure type.
- typedef struct pthread_permitnc_hook_s pthread_permitnc_hook_t

  The hook data structure type.

Enumerations

- enum pthread_permit_hook_type_t

  The type of hook.

Functions

- int pthread_permitc_pushhook_np (pthread_permitc_t *permit, pthread_permit_hook_type_t type, pthread_permitc_hook_t *hook)

  Pushes a hook.
- int pthread_permitnc_pushhook_np (pthread_permitnc_t *permit, pthread_permit_hook_type_t type, pthread_permitnc_hook_t *hook)

  Pushes a hook.
- pthread_permitc_hook_t * pthread_permitc_pophook_np (pthread_permitc_t *permit, pthread_permit_hook_type_t type)

  Pops a hook.
- pthread_permitnc_hook_t * pthread_permitnc_pophook_np (pthread_permitnc_t *permit, pthread_permit_hook_type_t type)

  Pops a hook.

5.2.1 Detailed Description

The hooks of pthread_permitc_t and pthread_permitnc_t. Consuming and non-consuming POSIX threads permit objects can have their operations upcalled to interested parties. This is used to provide useful extensions such as kernel file descriptor state mirroring. The following hooks are available:

- PTHREAD_PERMIT_HOOK_TYPE_DESTROY: Called just before a permit is destroyed.
- PTHREAD_PERMIT_HOOK_TYPE_GRANT: Called just after a permit is granted, but before waiters are woken.
- PTHREAD_PERMIT_HOOK_TYPE_REVOKE: Called just after a permit is revoked.

pthread_permit_hook_t_np is a structure defined as follows:

```
typedef struct pthread_permit_hook_s
{
  int (*func)(pthread_permitX_t permit, pthread_permit_hook_t_np *hookdata);
  void *data;
  pthread_permit_hook_t_np *next;
} pthread_permit_hook_t_np;
```

You almost certainly want to allocate this structure statically as no copy is taken. *func* should have the following form:

```
int pthread_permit_hook_grant(pthread_permit_hook_type_t
       type, pthread_permitX_t permit, pthread_permit_hook_t_np *hookdata)
{
  // Use hookdata->data as you see fit
  ...
  // Call previous hooks
  return hookdata->next ? hookdata->next->func(type, permit, hookdata->next) :
      0;
}
```

To add a hook, pthread_permitc_pushhook() and pthread_permitnc_pushhook() pushes a hook to the top of the call stack (i.e. is called first) by setting its *next* member to the previous top hook. pthread_permitc_pophook() and pthread_permitnc_pophook() delink the top hook and return it.

5.3 Permit initialisation

Initialises a permit.

Functions

- int pthread_permit1_init (pthread_permit1_t *permit, _Bool initial)

  Initialises a pthread_permit1_t.
- int pthread_permitc_init_np (pthread_permitc_t *permit, _Bool initial)

  Initialises a pthread_permitc_t.
- int pthread_permitnc_init_np (pthread_permitnc_t *permit, _Bool initial)

  Initialises a pthread_permitnc_t.

5.3.1 Detailed Description

Initialises a permit.

Returns

0: success; EINVAL: bad/incorrect permit.

5.4 Permit destruction

Destroys a permit.

Functions

- void pthread_permit1_destroy (pthread_permit1_t *permit)

  Destroys a pthread_permit1_t.
- void pthread_permitc_destroy_np (pthread_permitc_t *permit)

  Destroys a pthread_permitc_t.
- void pthread_permitnc_destroy_np (pthread_permitnc_t *permit)

  Destroys a pthread_permitnc_t.

5.4.1 Detailed Description

Destroys a permit.

5.5 Permit granting

Grants a permit.

Functions

- int pthread_permit1_grant (pthread_permitX_t permit)

  Grants a pthread_permit1_t.
- int pthread_permitc_grant_np (pthread_permitX_t permit)

  Grants a pthread_permitc_t.
- int pthread_permitnc_grant_np (pthread_permitX_t permit)

  Grants a pthread_permitnc_t.

5.5.1 Detailed Description

Grants a permit.

Returns

0: success; EINVAL: bad/incorrect permit.

Grants permit to one waiting thread. If there is no waiting thread, permits the next thread to wait.

If the permit is consuming (pthread_permit1_t and pthread_permitc_t), the permit is atomically transferred to the winning thread.

If the permit is non-consuming (pthread_permitnc_t), the permit is still atomically transferred to the winning thread, but the permit is atomically regranted. You are furthermore guaranteed that exactly every waiter at the time of grant will be released before the grant operation returns. Note that because of this guarantee, only one grant may occur at any one time per permit instance i.e. grant is a critical section. This implies that if a grant is operating, any new waits hold until the grant completes.

The parameter of the grant functions is a pthread_permitX_t which denotes any of pthread_permit1_t, pthread_-permitc_t and pthread_permitnc_t. The API of the permit grant has been so unified to allow the following idiom:

```
int ask_3rd_party_library_to_do_an_asynchronous_job(..., pthread_permitX_t
      permit, pthread_permitX_grant_func completionroutine);
...
pthread_permitnc_t permit;
pthread_permitnc_init(&permit);
ask_3rd_party_library_to_do_an_asynchronous_job(..., &permit,
      pthread_permitnc_grant);
...
// Wait for completion
pthread_permitnc_wait(&permit);
```

In other words, client code can supply whichever type of permit it prefers to third party library code. That third party library simply calls the supplied completion routine, thus granting the permit.

5.6 Permit revoking

Revokes a permit.

Functions

- void pthread_permit1_revoke (pthread_permit1_t *permit)

  Revokes a pthread_permit1_t.
- void pthread_permitc_revoke_np (pthread_permitc_t *permit)

  Revokes a pthread_permitc_t.
- void pthread_permitnc_revoke_np (pthread_permitnc_t *permit)

  Revokes a pthread_permitnc_t.

5.6.1 Detailed Description

Revokes a permit. Revoke any outstanding permit, causing any subsequent waiters to wait.

5.7 Permit waiting

Waits on a permit.

Functions

- int pthread_permit1_wait (pthread_permit1_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permit1_t.
- int pthread_permitc_wait_np (pthread_permitc_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permitc_t.
- int pthread_permitnc_wait_np (pthread_permitnc_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permitnc_t.
- int pthread_permit1_timedwait (pthread_permit1_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permit1_t for a time.
- int pthread_permitc_timedwait_np (pthread_permitc_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permitc_t for a time.
- int pthread_permitnc_timedwait_np (pthread_permitnc_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permitnc_t for a time.
- int pthread_permit_select_np (size_t no, pthread_permitX_t *permits, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on many permits.

5.7.1 Detailed Description

Waits on a permit.

Returns

0: success; EINVAL: bad permit, mutex or timespec; ETIMEDOUT: the time period specified by ts expired.

Waits for permit to become available, atomically unlocking the specified mutex when waiting. If mtx is NULL, never sleeps instead looping forever waiting for permit. If ts is NULL, returns immediately instead of waiting.

5.7.2 Function Documentation

5.7.2.1 int pthread_permit_select_np (size_t *no*, pthread_permitX_t * *permits*, pthread_mutex_t * *mtx*, const struct timespec * *ts*)

Waits on many permits.

Returns

0: success; EINVAL: bad permit, mutex or timespec; ETIMEDOUT: the time period specified by ts expired.

Waits for a time for any permit in the supplied list of permits to become available, atomically unlocking the specified mutex when waiting. If mtx is NULL, never sleeps instead looping forever waiting for a permit. If ts is NULL, waits **forever** (rather than return instantly).

On exit, if no error the permits array has all ungranted permits zeroed. Only the first granted permit is ever returned, so all other elements will be zero.

On exit, if error then only errored permits are zeroed. In this case many elements can be returned.

Note that the permit array you supply may contain null pointers - if so, these entries are ignored. This allows a convenient "rinse and repeat" idiom.

The complexity of this call is O(no). If we could use dynamic memory, or had OS support, we could achieve O(1).

5.8 Permit kernel object association

Associates a non-consuming permit with a kernel object's state.

Typedefs

- typedef struct pthread_permitnc_association_s * pthread_permitnc_association_t

  The type of a permit association handle.

Functions

- pthread_permitnc_association_t pthread_permitnc_associate_fd_np (pthread_permitnc_t *permit, int fds[2])

  Associates the state of a kernel file descriptor with the state of a pthread_permitnc_t.
- void pthread_permitnc_deassociate_np (pthread_permitnc_t *permit, pthread_permitnc_association_t assoc)

  Deassociates the state of a kernel file descriptor with the state of a pthread_permitnc_t.
- pthread_permitnc_association_t pthread_permitnc_associate_winhandle_np (pthread_permitnc_t *permit, HANDLE h)

  Associates the state of a Windows kernel file handle with the state of a pthread_permitnc_t.
- pthread_permitnc_association_t pthread_permitnc_associate_winevent_np (pthread_permitnc_t *permit, HANDLE h)

  Associates the state of a Windows kernel event handle with the state of a pthread_permitnc_t.

5.8.1 Detailed Description

Associates a non-consuming permit with a kernel object's state. Sets a file descriptor whose signalled state should match the permit's state i.e. the descriptor has a single byte written to it to make it signalled when the permit is granted. When the permit is revoked, the descriptor is repeatedly read from until it is non-signalled. Note that this mechanism uses the non-portable hook system but it does upcall previously installed hooks. Note that these functions use malloc().

Note that these calls are not thread safe, so do **not** call them upon a permit being used (i.e. these are suitable for initialisation and destruction stages only). Note you must call pthread_permit_deassociate() on the returned value before destroying its associated permit else there will be a memory leak.

For pthread_permit_associate_fd(), the int fds[2] are there to mirror the output from the pipe() function as that is the most likely usage. fds[0] is repeatedly read from until empty by revocation, while fds[1] has a byte written to it by granting. One can set both members to the same file descriptor if desired.

On Windows only, pthread_permit_associate_winhandle_np() is the Windows equivalent of pthread_permit_associate_fd(). For convenience there is also a pthread_permit_associate_winevent_np() which is probably much more useful on Windows.

Chapter 6

Data Structure Documentation

6.1 pthread_permitc_hook_t Struct Reference

The hook data structure.

`#include <pthread_permit.h>`

6.1.1 Detailed Description

The hook data structure.

Definition at line 257 of file pthread_permit.h.

The documentation for this struct was generated from the following file:

- pthread_permit.h

6.2 pthread_permitnc_hook_t Struct Reference

The hook data structure.

`#include <pthread_permit.h>`

6.2.1 Detailed Description

The hook data structure.

Definition at line 264 of file pthread_permit.h.

The documentation for this struct was generated from the following file:

- pthread_permit.h

Chapter 7

File Documentation

7.1 pthread_permit.h File Reference

Defines and declares the API for POSIX threads permit objects.

Data Structures

- struct pthread_permitc_hook_t

  The hook data structure.
- struct pthread_permitnc_hook_t

  The hook data structure.

Macros

- #define PTHREAD_PERMIT_APIEXPORT extern

  Adjust this to set how extern API is exported. NOTE: defining _USRDLL does the right thing on GCC, Clang and MSVC.
- #define PTHREAD_PERMIT_MANGLEAPI(api) pthread_##api##_np

  Adjust this to change extern API prefix and postfix. By default set to "pthread_" and "_np" respectively.
- #define PTHREAD_PERMIT_MANGLEAPINP(api) pthread_##api##_np

  Adjust this to change non-portable extern API prefix (e.g. Windows-only APIs). By default set to "pthread_".

Typedefs

- typedef struct pthread_permit1_s pthread_permit1_t

  The simplest and fastest implementation of a POSIX threads permit. Always consuming, non-hookable and non-selectable.
- typedef struct pthread_permitc_s pthread_permitc_t

  A consuming POSIX threads permit. Hookable and selectable.
- typedef struct pthread_permitnc_s pthread_permitnc_t

  A non-consuming POSIX threads permit. Hookable and selectable.
- typedef void * pthread_permitX_t

  A pointer to any of pthread_permit1_t, pthread_permitc_t and pthread_permitnc_t.
- typedef int(* pthread_permitX_grant_func)(pthread_permitX_t)

  A permit grant function prototype for any of the permit grant functions.
- typedef struct pthread_permitc_hook_s pthread_permitc_hook_t

The hook data structure type.

- typedef struct pthread_permitnc_hook_s pthread_permitnc_hook_t

  The hook data structure type.

- typedef struct pthread_permitnc_association_s * pthread_permitnc_association_t

  The type of a permit association handle.

Enumerations

- enum pthread_permit_hook_type_t

  The type of hook.

Functions

- int pthread_permitc_pushhook_np (pthread_permitc_t *permit, pthread_permit_hook_type_t type, pthread_permitc_hook_t *hook)

  Pushes a hook.

- int pthread_permitnc_pushhook_np (pthread_permitnc_t *permit, pthread_permit_hook_type_t type, pthread_permitnc_hook_t *hook)

  Pushes a hook.

- pthread_permitc_hook_t * pthread_permitc_pophook_np (pthread_permitc_t *permit, pthread_permit_hook_type_t type)

  Pops a hook.

- pthread_permitnc_hook_t * pthread_permitnc_pophook_np (pthread_permitnc_t *permit, pthread_permit_hook_type_t type)

  Pops a hook.

- int pthread_permit1_init (pthread_permit1_t *permit, _Bool initial)

  Initialises a pthread_permit1_t.

- int pthread_permitc_init_np (pthread_permitc_t *permit, _Bool initial)

  Initialises a pthread_permitc_t.

- int pthread_permitnc_init_np (pthread_permitnc_t *permit, _Bool initial)

  Initialises a pthread_permitnc_t.

- void pthread_permit1_destroy (pthread_permit1_t *permit)

  Destroys a pthread_permit1_t.

- void pthread_permitc_destroy_np (pthread_permitc_t *permit)

  Destroys a pthread_permitc_t.

- void pthread_permitnc_destroy_np (pthread_permitnc_t *permit)

  Destroys a pthread_permitnc_t.

- int pthread_permit1_grant (pthread_permitX_t permit)

  Grants a pthread_permit1_t.

- int pthread_permitc_grant_np (pthread_permitX_t permit)

  Grants a pthread_permitc_t.

- int pthread_permitnc_grant_np (pthread_permitX_t permit)

  Grants a pthread_permitnc_t.

- void pthread_permit1_revoke (pthread_permit1_t *permit)

  Revokes a pthread_permit1_t.

- void pthread_permitc_revoke_np (pthread_permitc_t *permit)

  Revokes a pthread_permitc_t.

- void pthread_permitnc_revoke_np (pthread_permitnc_t *permit)

  Revokes a pthread_permitnc_t.

- int pthread_permit1_wait (pthread_permit1_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permit1_t.
- int pthread_permitc_wait_np (pthread_permitc_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permitc_t.
- int pthread_permitnc_wait_np (pthread_permitnc_t *permit, pthread_mutex_t *mtx)

  Waits on a pthread_permitnc_t.
- int pthread_permit1_timedwait (pthread_permit1_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permit1_t for a time.
- int pthread_permitc_timedwait_np (pthread_permitc_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permitc_t for a time.
- int pthread_permitnc_timedwait_np (pthread_permitnc_t *permit, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on a pthread_permitnc_t for a time.
- int pthread_permit_select_np (size_t no, pthread_permitX_t *permits, pthread_mutex_t *mtx, const struct timespec *ts)

  Waits on many permits.
- pthread_permitnc_association_t pthread_permitnc_associate_fd_np (pthread_permitnc_t *permit, int fds[2])

  Associates the state of a kernel file descriptor with the state of a pthread_permitnc_t.
- void pthread_permitnc_deassociate_np (pthread_permitnc_t *permit, pthread_permitnc_association_t assoc)

  Deassociates the state of a kernel file descriptor with the state of a pthread_permitnc_t.
- pthread_permitnc_association_t pthread_permitnc_associate_winhandle_np (pthread_permitnc_t *permit, HANDLE h)

  Associates the state of a Windows kernel file handle with the state of a pthread_permitnc_t.
- pthread_permitnc_association_t pthread_permitnc_associate_winevent_np (pthread_permitnc_t *permit, HANDLE h)

  Associates the state of a Windows kernel event handle with the state of a pthread_permitnc_t.

7.1.1 Detailed Description

Defines and declares the API for POSIX threads permit objects.

Definition in file pthread_permit.h.

Index