| | |
|---|---|
| Document number: | P0320R1 |
| Date: | 2016-10-12 |
| Project: | ISO/IEC JTC1 SC22 WG21 Programming Language C++ |
| Audience: | Library Evolution Working Group/Concurrency Working Group |
| Reply-to: | Vicente J. Botet Escribá <vicente.botet@nokia.com> |

# Thread Constructor Attributes

**Abstract**

This paper presents an extension of `thread` construction allowing to pass an additional implementation defined attributes parameter.

# Table of Contents



# History

## Revision 1

The 1st revision of this proposal fixes some typos and takes in account the feedback from Oulu meeting. Next follows the direction of the committee:

- Make `std::thread::attributes` implementation defined.

# Introduction

Depending on the platform, there are some attributes that can be provided at construction time. However these attributes are platform dependent, and while there are some such as the stack size that are quite current on major operating systems such as Posix, the stack size of individual threads tends to be fixed at thread creation time, some platforms don't know what to do with this stack size, e.g. platforms with virtual / abstract machine.

This paper presents an extension of `thread` construction allowing to pass an additional implementation defined attributes parameter.

# Motivation and Scope

Today we can construct an instance of `thread` with a function or callable object , e.g:

```
void find_the_question(int the_answer);

std::thread deep_thought_2(find_the_question, 42);
```

Threads launched in this way are created with implementation defined thread attributes as stack size, scheduling, priority, ... or any platform specific attributes.

However in some specific domains it is important to be able to be more specific so that the resources are used in an optimal way.

As each platform has its own specific thread construction attributes, it is not evident how to provide a portable interface that allows the user to set the platform specific attributes. This paper stay in the middle road through the class `std::thread::attributes` which allows to set at least in a non portable way the platform specific attributes.

It is up to each implementation to provide the specific interface for each platform. Whether several implementation provide the same interface fro a specific platform is out of the scope of this proposal.

# Proposal

This paper proposes then to

- add a implementation dependent `thread::attributes` class.
- add `thread` constructors taking a `thread::attributes` parameter.

## How to set the stack size?

In an implementation providing a `std::thread::attributes::set_stack_size` function, the stack size attribute of a thread can be set as follows:

```
std::thread::attributes attrs;
attrs.set_stack_size(4096*10);
std::thread deep_thought_2(attrs, find_the_question, 42);
```

Even for this simple attribute there are portable issues as some platforms could require that the stack size should have a minimal size and/or be a multiple of a given page size. It is up to the library implementation to define its interface and possibly adapt the requested size to the platform constraints so that the user doesn't need to take care of it.

## Using a `native_handle_type native_handle()`

The implementation can provide also function to get a native handle `native_handle_type native_handle()`. E.g. on Posix platforms the user will need to get the thread attributes native handle and use it for the appropriate attribute.

Next follows how the user could set the stack size and the scheduling policy on Posix platforms.

```
std::thread::attributes attrs;
// ... pthread version
pthread_attr_setschedpolicy(attr.native_handle(), SCHED_RR);
pthread_attr_setstacksize(attr.native_handle(), 4096*10);
std::thread th(attrs, find_the_question, 42);
```

## Implementation defined `thread::attributes` interface

On Windows platforms it is not so simple as there is no type that compiles the thread attributes. There is one attribute linked to the creation of a thread on Windows that is emulated via the `thread::attributes` class, this is the `LPSECURITY_ATTRIBUTES lpThreadAttributes`. The implementation can provide a non portable `set_security` function so that the user can provide it before the thread creation as follows

```
std::thread::attributes attrs;
// set non portable attribute stack_size
attr.set_stack_size(4096*10);
// set non portable attribute security
LPSECURITY_ATTRIBUTES sec;
attr.set_security(sec); // non portable
std::thread th(attrs, find_the_question, 42);
//...
```

# Design rationale

## Why `std::thread::attributes` implementation defined?

There are no single thread attribute that can be implemented in a portable way. Some platforms allow to set the stack size, other have two stacks, other allows to set the thread name, ... Saying that C++ has a stack traditionally opens up a can of worms.

Letting the `std::thread::attributes` implementation defined allow an implementation to provide the best interface for a specific platform.

## Single `std::thread` constructor with a `std::thread::attributes` versus specific `std::thread` constructors

SG14 presents an approach adding an additional `std::thread` constructors taking a `required_stack_size` parameter.

Independently of the portability of this attribute, this proposal prefers to let open the interface for other attributes and store them in a specific implementation defined class `std::thread::attributes` .

## Checking for `thread::attributes` value

Some Posix let the user retrieve the value of an attribute Posix.pthreadgetattrnp. On this systems, the user could check the value of an attribute using the thread native handle.

We could provide a way to get these attributes.

This paper doesn't propose yet an interface to get the attributes, but if there is an interest, we could add a

function to the thread class.

```
thread::current_attributes thread::get_attributes() const;
```

```
namespace this_thread {
    thread::current_attributes get_attributes() const;
}
```

Note that `thread::current_attributes` would need to be a different class than `thread::attributes` as we don't want them to be modifiable.

Note that in such Posix systems, the user can already get the native handle of a `std::thread` or of the native handle current thread and get then the attributes.

Returning it by value ensures that the library don't need to store anything in addition to what the platform stores.

# Proposed wording

The wording is relative to the C++ standard working draft N4594.

## Thread library

**Update Class thread [thread.thread.class] section with**

### Class thread [thread.thread.class]

```
namespace std {
namespace experimental {
inline namespace concurrency_v3 {

  class thread {
  public:
    // add after id
    class attributes; // implementation defined

    // add after thread constructor
    template <class F, class ...Args>
    explicit thread(attributes const& attr, F&& f, Args&&... args);

  };

}}}
```

### Class `thread::attributes`

```
namespace std {
namespace experimental {
inline namespace concurrency_v3 {

class thread::attributes;

}}}
```

Implementations are free to provide the interface of this class in a non-portable way.

#### Update thread constructors [thread.thread.constr] adding

```
template <class F, class ...Args> explicit thread(F&& f, Args&&... args);
template <class F, class ...Args> explicit thread(attributes const&, F&& f, Args&&..
```

As before

*Remarks*: The first overload constructor shall not participate in overload resolution if `decay_t<F>` is the same type as `std::thread` or `std::thread::attributes`.

*Effects*: Constructs an object of type `thread`, taking in account the passed attributes. The first overload behaves as if a default `thread::attributes` parameter was passed. ....

# Implementability

This proposal can be implemented as pure library extension, without any compiler magic support. Boost.Thread provides it since version 1.51

# Open points

The authors would like to have an answer to the following points if there is at all an interest in this proposal:

- Do we want a way to get the current attributes of a thread?
- Do we want this for the IS or for the Concurrent TS?

# Acknowledgements

Thanks to all that commented this proposal helping me to improve the paper as a whole.

Thanks for the feedback from the WG1, making this proposal much simpler.

Special thanks and recognition goes to Technical Center of Nokia - Lannion for supporting in part the production of this proposal.

# References

- N4594 Working Draft, Standard for Programming Language C++ http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/n4594.pdf
- P0159R0 P0159 - Draft of Technical Specification for C++ Extensions for Concurrency http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0159r0.html
- Boost.Thread http://www.boost.org/doc/libs/1600/doc/html/thread.html
- SG14 Controlling Thread Stack Size at Creation Time http://hdeb.clg.qc.ca/WG21/SG14/threadctorstack_size.pdf
- Windows CreateThread() https://msdn.microsoft.com/enus/library/windows/desktop/ms682453%28v=vs.85%29.aspx
- Posix pthreadcreate() http://pubs.opengroup.org/onlinepubs/007908775/xsh/pthreadcreate.html
- Posix.pthreadgetattrnp pthreadgetattrnp() http://man7.org/linux/man-pages/man3/pthreadgetattrnp.3.html