Controlling Thread Stack Size at Creation Time

Document #: Nxxxx

Date: 2015-09-22

Revises: n/a

Reply to: Patrice.Roy@USherbrooke.ca

Contents

Abstract

Low-latency systems need to fix some thread characteristics at thread creation time. The standard threading API provided for C++ does not currently provide a way to do this.

With the current proposal, we seek to address this problem.

Background

In low-latency systems, and more precisely in video games development, maintaining fine-grained control over a program's resources is essential. In many areas, the C++ language and its standard library provide programmers with the level of control they need.

There are however some cases where the API provided by the standard library is lacking. In some of those cases, the impact of those missing elements can go as far as preventing the adoption of standard utilities by specialized users, forcing them to rely on platform-specific utilities instead. This is a disservice to everyone.

This proposal aims to cover one such missing part: the possibility to specify the stack size of a thread at construction time.

The Problem

The videogame industry typically uses threads with specific stack sizes, and these sizes often differ from the operating system default. As videogames often target consoles, which are essentially high-end embedded systems, game programmers can have intimate knowledge of the resources available to their application and are able to determine their needs with more precision that programmers for more general platforms and environments could.

In some game companies, adoption of standard threading facilities has been hampered due to the lack of functionalities allowing control over the characteristics of threads that would be specified at construction time. Indeed, on major operating systems such as [1][2], the stack size of individual threads tends to be fixed at thread creation time.

Benefits

The benefits that would come from solving this problem are a wider adoption of the standard threading facilities of C++, or at least the removal of a significant barrier to this adoption. In the author's opinion, this is a desirable outcome.

Prior Art

The standard threading facilities offer access to the underlying platform native handles. This allows programmers to use facilities not offered by the standard once a thread has been created. This facility, useful as it can be, require a thread to be created; as such, they do not address the need to customize those characteristics of a thread that have to be specified at thread creation time.

For the same reason, adding factory functions to combine thread creation and stack size enforcement would not be sufficient, at least not if said factory function was written strictly based on standard library utilities[1].

Taking for example two well-known platforms, we can show the need for a thread creation time facility in order to solve the control over stack size problem.

On Microsoft Windows, using the Win32 API [1], it is possible to specify at thread creation time:

- Operating system-specific security attributes
- Thread stack size
- A specific start address, and
- The fact that the thread should start executing right away or be created in a suspended, to be resumed execution state

On operating systems that support the pthread API provided for POSIX, it is possible to specify a number of parameters at thread creation time [2], including stack size through the pthread_attr_setstacksize() [3] operation of a pthread_attr.

Boundaries of this proposal

This proposal aims to cover the stack size question. The start-as-suspended case can be implemented manually through condition variables and fire-and-forget signals implanted through std::promise and std::future<void>[4], and as such does not seem as urgent.

---

[1] It could be done if the factory function encapsulated native thread creation and parameterization before constructing the standard thread object, of course.

Addressing a complete set of platform-specific considerations would require a more thorough examination in order to be addressed in a satisfactory manner.

Proposed Wording

In §30.3.1, [thread.thread.class], insert the following additions:

```
namespace std {
  class thread {
  public:
    // types:
    class id;
    typedef implementation-defined native_handle_type;
    class required_stack_size {
      explicit required_stack_size(size_t);
      size_t value() const noexcept;
      // …
    };
    // construct/copy/destroy:
    thread() noexcept;
    template <class F, class ...Args> explicit thread(F&& f, Args&&... args);
    template <class F, class ...Args> explicit thread(required_stack_size,F&& f, Args&&... args);
    // …
```

Insert the following section before §30.3.1.2 [thread.thread.constr]:

30.3.2.1 Class thread::required_stack_size [thread.thread.reqstack]

1 An object of type thread::required_stack_size represents an explicit stack size requirement for thread of execution. It shall store the value passed to its constructor internally.

2 thread::required_stack_size shall be a trivially copyable class (Clause 9).

size_t value() noexcept;

3 Effects: returns the value that was passed to the object when it was constructed.

After ~~§30.3.1.2~~ §30.3.1.3 [thread.thread.constr], paragraph 9, perform the following insertions:

template <class F, class ...Args> explicit thread(required_stack_size sz, F&& f, Args&&... args);

10 Requires: F and each Ti in Args shall satisfy the MoveConstructible requirements. INVOKE (DECAY_COPY (std::forward<F>(f)), DECAY_COPY (std::forward<Args>(args))...) (20.9.2) shall be a valid expression.

11 Remarks: This constructor shall not participate in overload resolution if decay_t<F> is the same type as std::thread.

13 Remarks: This constructor shall not participate in overload resolution if decay_t<F> is the same type as std::thread.

14 Effects: Constructs an object of type thread. The new thread of execution executes INVOKE (DECAY_COPY (std::forward<F>(f)), DECAY_COPY (std::forward<Args>(args))...) with the calls to DECAY_COPY being evaluated in the constructing thread. Any return value from this invocation is ignored. [Note: This implies that any exceptions not thrown from the invocation of the copy of f will be thrown in the constructing thread, not the new thread. —end note] If the invocation of INVOKE (DECAY_COPY (std::forward<F>(f)), DECAY_COPY (std::forward<Args>(args))...) terminates with an uncaught exception, std::terminate shall be called.

15 The associated thread of execution shall have a stack size of sz.value().

16 Synchronization: The completion of the invocation of the constructor synchronizes with the beginning of the invocation of the copy of f.

17 Postconditions: get_id() != id(). *this represents the newly started thread.

18 Throws: system_error if unable to start the new thread.

19 Error conditions:

— resource_unavailable_try_again — the system lacked the necessary resources to create another thread, or the system-imposed limit on the number of threads in a process would be exceeded.

Alternative Approaches

There are at least two other approaches to solve the problem addressed in this proposal:

- One would be to allow programmers to use platform-specific services to create threads, then pass the resulting native handle to a thread object. This does not lead to portable client code, however.
- Another would be to enhance the startup argument type to make it extensible or adaptable in such a way as to allow specifying more characteristics of interest at thread

creation time. A summary examination of platform-specific APIs does not convince the author that a more extensible implementation would be fruitful in this case.

Finally, the name std::thread::required_stack_size is a suggestion; should the committee prefer a different naming scheme, I do not foresee resistance on my part.

Acknowledgements

Thanks to many programmers from various game programming studios, including WB Games Montréal, Ubisoft Montréal and EIDOS/ Square Enix Montréal, for having brought this problem to my attention.

References

[1] CreateThread() : https://msdn.microsoft.com/en-us/library/windows/desktop/ms682453%28v=vs.85%29.aspx

[2] pthread_create(): http://pubs.opengroup.org/onlinepubs/007908775/xsh/pthread_create.html

[3] pthread_attr_setstacksize(): http://man7.org/linux/man-pages/man3/pthread_attr_setstacksize.3.html

[4] Meyers, S., Effective Modern C++, O'Reilly Media (2015), Item 39

Revision History

n/a