Symantec™

Thread Attributes as Designators

Zhihao Yuan

2024/3/18

zhihao.yuan@broadcom.com

Proposal

Symantec

```
std::thread thr(
    {
        .name = "worker",
        .stack_size_hint = 16384,
    },
    [] { std::puts("standard"); });
```

Extensions

Symantec

```
std::thread thr(
    __gnu_cxx::posix_thread_attributes{
        .schedpolicy = SCHED_FIFO,
    },
    [] { std::puts("vendor extension"); });
```

How this is done

Symantec

```
class thread
{
  public:
    struct attributes
    {
        string const &name = {};
        size_t stack_size_hint = 0;
    };

    template<class F, class... Args> requires is_invocable_v<F, Args...>
    explicit thread(F &&, Args &&...);

    template<class A = attributes, class F, class... Args>
    requires is_invocable_v<F, Args...>
    explicit thread(A, F &&, Args &&...);
```

Same on jthread

Symantec

```
class jthread
{
  public:
    using attributes = thread::attributes;



    template<class F, class... Args> requires is_invocable_v<F, Args...>
    explicit jthread(F &&, Args &&...);

    template<class A = attributes, class F, class... Args>
    requires is_invocable_v<F, Args...>
    explicit jthread(A, F &&, Args &&...);
```

Specifying all standard thread attributes

Symantec

```
std::thread t1(
    {.name = std::format("worker {}", i), .stack_size_hint = 16384},
    [] { std::puts("everything"); });
```

Specifying only the thread name

Symantec

```
std::thread t2({.name = "gui"}, [] { std::puts("only name"); });
```

Specifying only the thread name, a different style


```
std::thread t2a({"gui"}, [] { std::puts("only name"); });
```

Providing only a hint to the stack size

Symantec

```
std::thread t3({.stack_size_hint = 4096}, std::puts, "only size");
```

Declaring the attributes object as a variable

Symantec

Not recommended

```
std::thread::attributes attrs{.name = std::format("worker {}", i)};
std::thread t4(attrs, std::puts, "lifetime extension");
```

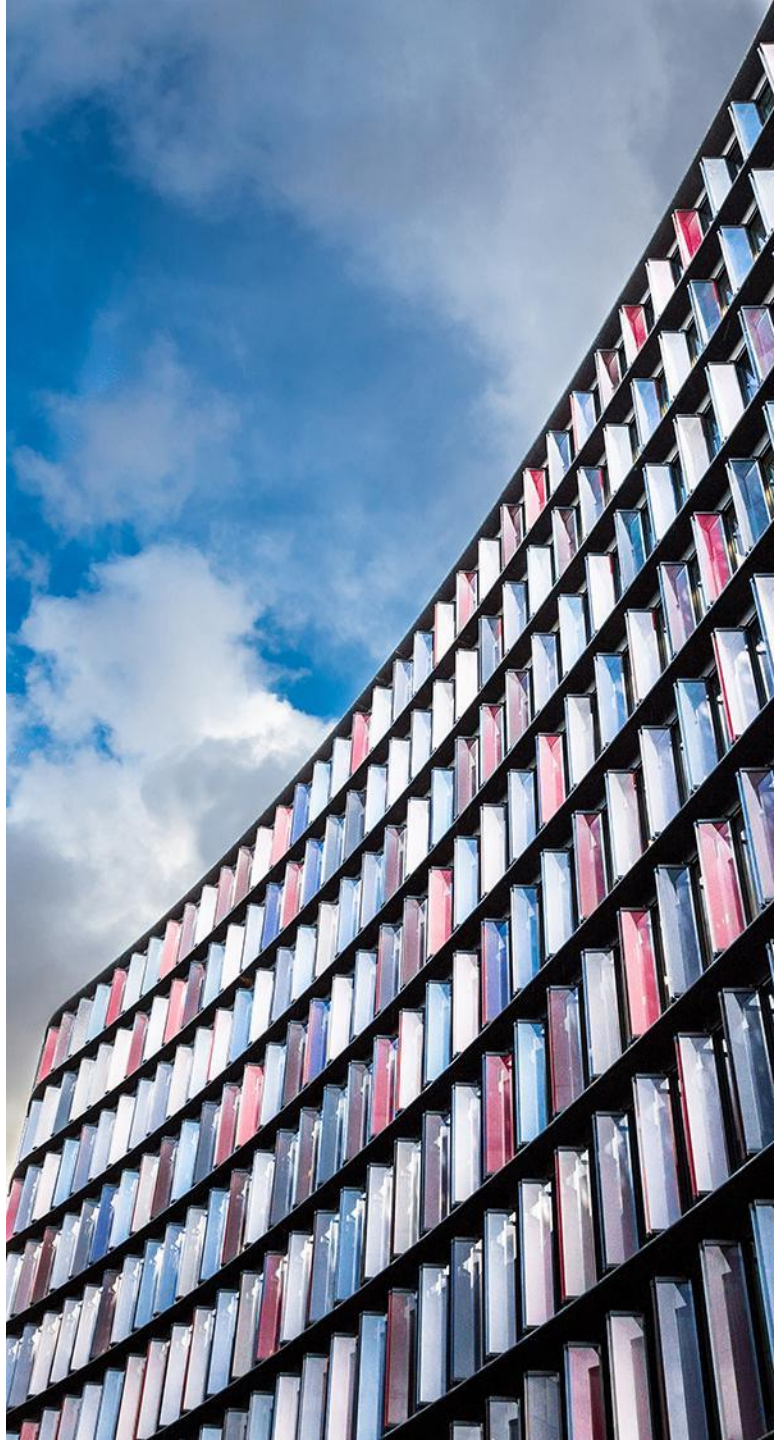
Substituting in non-standard thread attributes

Symantec

Class name is required

```
std::thread t5(
    __gnu_cxx::posix_thread_attributes{.schedpolicy = SCHED_FIFO},
    std::puts, "vendor extension");
```

Why not pass attributes individually



Is this required?

Symantec

```
struct posix_thread_attributes : std::thread::attributes
{
    int schedpolicy = SCHED_OTHER;
};
```

P2019R5

Symantec

```
std::thread thr(std::thread::stack_size_hint(16384),
                __gnu_cxx::posix_schedpolicy(SCHED_FIFO),
                [] { std::puts("you want them to work together"); });
```

Do `std::thread::stack_size_hint` and `__gnu_cxx::posix_schedpolicy` share the same guts?

PoC guts (from P2019)

Symantec

```
int __libcpp_thread_create(__libcpp_thread_t *__t, void *(*__func)(void *),
                           void *__arg, size_t __stack_size,
                           const __libcpp_threadname_char_t *__name)
{
    int res = 0;
    if (__stack_size != 0)
    {
        pthread_attr_t attr;
        res = pthread_attr_init(&attr);
        if (res != 0)
        {
            return res;
        }
        res = pthread_attr_setstacksize(&attr, __stack_size);
        res = pthread_create(__t, &attr, __func, __arg);
        res = pthread_attr_destroy(&attr);
    }
```

P2019R5

Symantec

```
std::thread thr(std::thread::stack_size_hint(16384),
                __gnu_cxx::posix_schedpolicy(SCHED_FIFO),
                [] { std::puts("you want them to work together"); });
```

Do `std::thread::stack_size_hint` and `__gnu_cxx::posix_schedpolicy` share the same guts?

Do `std::thread::name_hint` and `__gnu_cxx::posix_name` accept the same types?

Symantec

std::thread::name_hint PoC

```
template<typename T>
class name_hint
{
    basic_string_view<T> __name;

  public:
    explicit
      name_hint(basic_string_view<T>);
    name_hint(name_hint &&) = delete;
    name_hint(const name_hint &) = delete;
};
```

Hypothetical posix_name

```
template<size_t N> requires(N <= 16)
class posix_name
{
    char __name[N];

  public:
    posix_name(char const (&name)[N]);
};
```

P2019R5

Symantec

```
std::thread thr(std::thread::stack_size_hint(16384),
                __gnu_cxx::posix_schedpolicy(SCHED_FIFO),
                [] { std::puts("you want them to work together"); });
```

Do `std::thread::stack_size_hint` and `__gnu_cxx::posix_schedpolicy` share the same guts?

Do `std::thread::name_hint` and `__gnu_cxx::posix_name` accept the same types?

Do `std::thread::name_hint` and `__msvc::threadname_info` have the same effect?

Symantec

std::thread::name_hint PoC

```
int n = MultiByteToWideChar(CP_UTF8, 0, __name.data(),
                            -1, nullptr, 0);
auto buf = std::make_unique_for_overwrite<wchar_t[]>(n);
MultiByteToWideChar(CP_UTF8, 0, __name.data(), -1,
                    buf.get(), n);
SetThreadDescription(__thread_id, buf.get());
```

Hypothetical __msvc::threadname_info

```
THREADNAME_INFO info{
    .dwType = 0x1000,
    .szName = __name.data(),
    .dwThreadID = __thread_id,
};
__try
{
    RaiseException(MS_VC_EXCEPTION, 0,
                   sizeof(info) / sizeof(ULONG_PTR),
                   (ULONG_PTR *)&info);
}
__except (EXCEPTION_EXECUTE_HANDLER)
{}
```

P3072R2

Symantec

```
struct posix_thread_attributes
{
    char const *name = nullptr;
    size_t stacksize = 0;
    int schedpolicy = SCHED_OTHER;
};
```

`__gnucxx::posix_thread_attributes` and `std::thread::attributes` may use different guts

`__gnucxx::posix_thread_attributes::stacksize` and `std::thread::attributes::stack_size_hint` may use different names

`__gnucxx::posix_thread_attributes::name` and `std::thread::attributes::name` may be of different types, may even have different effects

Why not char *, string_view, or string

Symantec™

C-style

Symantec

```
struct attributes
{
    const char *name = nullptr;
    size_t stack_size_hint = 0;
};
```

- × Verbose to call
- × Ask for dangling

```
std::thread::attributes attrs{.name = std::format("worker {}", i).c_str()};
```

string_view

Symantec

```
struct attributes
{
    string_view name = {};
    size_t stack_size_hint = 0;
};
```

- × Still dangle
- × More work if input is not null-terminated

```
std::thread::attributes attrs{.name = std::format("worker {}", i)};
```

std::string

Symantec

```
struct attributes
{
    string name = {};
    size_t stack_size_hint = 0;
};
```

- ✓ Does not dangle
- ✓ Null-terminated

```
std::thread::attributes attrs{.name = std::format("worker {}", i)};
```

std::string

Symantec

```
struct attributes
{
    string name = {};
    size_t stack_size_hint = 0;
};
```

- ✓ Does not dangle
- ✓ Null-terminated
- ✕ Easy to trigger a copy

```
auto launch_first(std::vector<std::string> const &names)
{
    return std::thread({.name = names.front()}, this);
}
```

const string&

Symantec

```
struct attributes
{
    string const &name = {};
    size_t stack_size_hint = 0;
};
```

- ✓ Null-terminated
- ✓ Do not dangle*

```
std::thread::attributes attrs{.name = std::format("worker {}", i)};
std::thread::attributes attrs = get_thread_attributes(); // dangle?
```

Dangles only if get_thread_attributes is plain wrong

const string&

Symantec

```
struct attributes
{
    string const &name = {};
    size_t stack_size_hint = 0;
};
```

- ✓ Null-terminated
- ✓ Do not dangle
- ✓ Reference semantics

```
auto launch_first(std::vector<std::string> const &names)
{
    return std::thread({.name = names.front()}, this);
}
```

Questions so far?