[[nodiscard]] Policy

https://wg21.link/p3162r0

Darius Neațu <dariusn@adobe.com>
David Sankel <dsankel@adobe.com>

2024-03-19 Tokyo WG21 Meeting

[[nodiscard]] **History**

- Proposal of [[unused]], [[nodiscard]] and [[fallthrough]] attributes (P0068R0), Andrew Tomazos
- Wording for [[nodiscard]] attributes (P0189R1), Andrew Tomazos
- [[nodiscard]] in the Library (P0600R1), Nicolai Josuttis

Nico's proposed placement

- For existing APIs:
  - not using the return value always is a “huge mistake” (e.g., always resulting in resource leak)
  - not using the return value is a source of trouble and easily can happen (not obvious that something is wrong)
- For new APIs (not been in the C++ standard yet):
  - not using the return value is usually an error.

Since then...

- Case-by-case LEWG debates
- Inconsistent `[[nodiscard]]` placement
- Users are perplexed on when to use the feature

Survey

Standard library instances

- `.empty()`
- `operator new` and `allocate()` functions
- `async()`
- `jthread::get_id()`
- NOT on `this_thread::get_id()`
- NOT on error types (e.g. `expected`, `error_code`)
- NOT on C allocation functions (e.g. `malloc`)
- SOMETIMES present on `operator==`

Clang Tidy

- modernize-use-nodiscard
  - Add `[[nodiscard]]` to non-void, non-template, const member functions that return.
- bugprone-unused-return-value
  - Specific functions (e.g. `isspace`, `lower_bound`)
  - Specific return types (e.g. `error_condition`, `expected`)

Important observations

`[[nodiscard]]` behavior not mandated in the library

- Compiler warnings not mandated in general
- As-if rule

`[[nodiscard]]` **in implementations**

- libstdc++ and Visual C++ make their own decisions
- libc++ mimics the standard placement

Other consequences of `[[nodiscard]]` placement

- Presence in https://cppreference.org function signatures and other training materials.
- This exposure impacts practice

Driving principles

1. Minimize complexity
2. Focus on the 90% use case
3. Center on outcomes

Minimize complexity

- Make code approachable to new users
- Reduce maintenance burden
- Improve longevity

Rules out placing `[[nodiscard]]` almost everywhere

Focus on the 90% use case

A handful of placements addresses the most severe bugs

Examples

```
std::vector<int> v{...};
v.empty();                  // Using 'clear' instead of 'empty' is a
                            // common bug, especially for those coming from
                            // another language.

std::unique_ptr<X> x{...};
x.release();                // Releasing the 'unique_ptr' in this example
                            // results in a memory leak.

std::async(job_x, &x, ...); // Accidentally ignoring the return value of
std::async(job_y, &y, ...); // async gives the false impression that jobs
                            // are run in parallel.

calloc(size * sizeof(int)); // Ignoring the return value of calloc is
                            // a memory leak.
```

Center on outcomes

- Vendors can do whatever they want, but…
- we should consider the larger impact of the decision

Our proposal

- Place `[[nodiscard]]` on functions where ignoring a return value is inevitably a severe defect, such as resource leakage.
- Place `[[nodiscard]]` on functions where overlooking the return value is a common mistake, such as function names frequently confused with others.
- Place `[[nodiscard]]` on types designed to communicate errors as function return values.