Memory allocation in safety critical code

International standard requirements

There are international standards that describe safety critical development process. ISO:26262 can be taken as the reference.

ISO:26262

1. No dynamic objects or variables, or else online test during their creation
2. 7.4.13 An upper estimation of required resources for the embedded software shall be made including:
    a. the execution time;
    b. the storage space;
    Example: RAM for stacks and heaps, ROM for program and non-volatile data.

Safety critical coding guidelines

MISRA C++:2008

1. Rule 18-4-1 Dynamic heap memory allocation shall be avoided
    - **Quote: "Dynamic heap memory allocation may lead to memory leaks, data inconsistency, memory exhaustion, non-deterministic behavior, etc."**

Adaptive Autosar C++14 (2018)

Challenges arising due to dynamic memory usage

- Memory leaks (Mitigation: Smart pointers, RAII)
- **Memory fragmentation** (Can be Mitigated: Custom allocator without memory fragmentation, but in fact is not addressed by the standard)
- Invalid memory access (Mitigation:C-style memory functions shall not be used)
- **Erroneous memory allocations** (There is no way to estimate and pre-allocate the required amount of dynamic memory except reverse engineering)
- **Not deterministic execution time of memory allocation and deallocation** (Can be mitigated: Custom allocator shall guarantee time constraints for memory allocation/deallocation but in fact is not addressed by the standard)

AUTOSAR guidelines suppose that "No dynamic objects or variables, or else online test during their creation" ISO:26262 requirement is addressed if the AUTOSAR rules, related to dynamic memory management, are fulfilled in the code.

- **Quote: “Dynamic memory allowed, but under multiple constraints affecting memory management. Allocated objects lifetime maintenance delegated to shared pointers and memory management objects. Restrictions set for custom implementations of dynamic memory allocation and placement new.”**

State of art in C++ standard

Currently there are two major ways for dynamic memory allocation in C++ standard library:

1. With Allocators (containers, other classes, e.g. std::shared_ptr)
2. Without Allocators. Allocation is hidden underneath for the particular functionality (e.g. in LLVM std::thread, std::sort, etc.).

Main issues

There are three major issues with dynamic memory allocation in C++:

1. There is no way to estimate and pre-allocate the required amount of dynamic memory. The issue is valid for both approaches (with allocator and without one)
    a. There is no memory guarantees/constraints defined in C++ that can be used by class/function
    b. Even if user has possibility to replace standard allocator by custom one there is no information about required memory by the particular container or allocator-aware object. Paper P1731 (see below) tries to address this issue. Note: For some classes (e.g. std::function, std::package_task) Allocator support was added for C++11 but was removed in C++17 (P0302) due to unaddressed open questions.
2. Impossibility to replace allocator for non-allocator-aware API.
    a. Developer may reverse engineer the implementation to figure out what functions are needed to be globally replaced (e.g. replacement of malloc) but it is not portable (different implementations may use different APIs. e.g. mmap)
    b. The allowed APIs for dynamic memory allocation and/or allocation strategy are not defined by the standard.
3. Non-deterministic execution time.
    a. Can be partially addressed by Custom Allocator for the allocator-aware-APIs but if user knows the allocation strategy (number of blocks, block size, etc.)
    b. Still cannot be addressed for non-allocator-aware API.

Current Proposals

P1731R1 Memory helper functions for Containers

Proof of concept:implementation done. Open questions/Next steps: Nested containers, Allocator interpretation of memory config.

Provides:

- API that takes allocator-aware class and the problem size (e.g. max size of the container)
- Returns memory config with the following information:
    - Concurrent memory blocks -> size_t (the number of simultaneously allocated blocks for which deallocate method is not called yet)
    - Memory block size -> size_t (the size of the one allocated block in bytes)
    - Memory block alignment -> size_t (alignment for each block)
    - Total amount of blocks -> size_t (calculates when it’s possible)
- Provides use-cases. This is public API that tells about usage intentions of allocator aware class (e.g user is not going to call shrink_to_fit and resize for std::vector) for better memory amount guarantees.