arenacall calling convention

1 Introduction

There has been much debate how to prevent memory leaks in C++, and what changes might be needed to comply with the executive order to use memory safe languages without reengineering the world in Rust.

The inspiration for this proposal is the legacy segment registers of Intel 80286 processors that provide unchecked access within a data segment but checked access to change DS register. Google Native Client (NaCl) used this technique to replace segment register instructions with checked equivalents to implement a sandbox.

The proposal is a half-way house between the current scheme where the heap has unchecked access and compiler-controlled pointers of Rust and C++ code analysis. It does so by standardizing Arena Allocators and adding a calling convention to pass a reference to the current threads arena.

2 Arena

Arena allocators are a common pattern for high-performance servers that need to ensure that memory allocated during a session is released at the end of the session without heap fragmentation or memory leaks.

An example of Arena allocator is provided by google protobuf (https://protobuf.dev/reference/cpp/arenas/) with a similar allocator for LevelDB/derivations and apr_bucket for Apache httpd.

Standardizing the different implementations of Arena allocation would be a worthy endeavor but does not justify the effort on its own but would for memory safety.

```
arena alloc = make_unique<std::Arena>();
```

would indicate that `alloc` is used as the default arena allocator while it remains in scope.

```
std::Arena stored = arena;
```

Would store the current arena for use by a std::allocator for collections that are part of a larger {session, singleton, configuration}.

3 arenacall

This calling convention would extend the AMD64 ABI (where 'this' is the first parameter), with the arena as the second parameter, or thiscall convention (where 'this' is passed in register) with a register reserved for the arena. The arena would be either the current arena of the caller or ZERO to indicate that no arena is passed.

arenacall functions for memory allocation and free would use the arena pointer for allocation and free if passed, and the CRT heap otherwise.

```
void* __arenacall operator new (size_t size);
void __arenacall operator delete (void* p);
void __arenacall operator delete[] (void* p);
void* __arenacall malloc(size_t size )
void __arenacall free( void* ptr );
```

arenacall functions would not need to pass the arena as a parameter, or use a coding convention to locate the arena. Protobuf provides a convention that service functions can call GetArena() to locate the active arena when allocating return messages. The function

```
unique_ptr<Value> RockSpace::Find(const FindRequest& request)
{
  unique_ptr<Value> result(Arena::CreateMessage<Value>(request.GetArena()));
  return move(result);
}
```

Could become:

```
unique_ptr<Value> __arenacall RockSpace::Find(const FindRequest& request)
{
  auto result = make_unique<Value>(); // uses arena
  return move(result);
}
```

Or (legacy code)

```
Value* __arenacall RockSpace::Find(const FindRequest& request)
{
  assert(arena);
  return new Value(); // uses arena, but doesn't leak outside arena scope
}
```

arenacall ensures that any function that allocates memory will use the current arena and will not leak memory after the arena is disposed, unless it calls standard functions that allocate memory on the heap.

Like checked iterators, a compiler directive like _CHECKED_ARENA=1 would allow functions like std::begin() and std::end() to check that arenacall functions are not referencing memory outside the bounds of {stack, this, arena}.

Setting the calling convention for the member, class or compiler parameter will enable arenacall with minimal changes to source code, and no logic change of algorithm.

4 Conclusion

- arenacall is suitable for scenarios where Rust might otherwise be considered, recognizing that Rust functions are normally called from C/C++, and call C/C++ foundation functions and cannot therefore guarantee safety.
- arenacall is suitable for messaging services that would otherwise use a framework specific allocator or arena.
- arenacall is suitable for scenarios where a system allows domain modules to be loaded but requires an elevated level of assurance that they cannot overwrite memory of other modules and do not leak memory.
- arenacall would be faster for methods that allocate many temporary objects –saving heap calls may offset the cost of checked arena assertions.
- arenacall embraces the C++ principle that you do not pay for options you do not choose to use.