

vollmann engineering gmbh

>

C++OS

Standard C++ as RTOS API

ESE

December 2022

Detlef Vollmann

vollmann engineering gmbh

vollmann engineering gmbh

>

C++OS

Standard C++ as RTOS API

Detlef Vollmann
vollmann engineering gmbh
Luzern, Switzerland

dv@vollmann.ch
<http://www.vollmann.ch/>



Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



What?

- C++ Standard Interface
 - not implementation
 - C++ compiler is not an OS
- C++OS
 - a specific implementation
 - some details



Why?

- Portability
- Testing / Simulation
 - on host
- Maintenance
 - well known API
- Control Structures
 - well known mechanisms



Overview

What and Why?

C++

execution

C++OS

Examples

C++OS Implementation

Why C++OS?

References



Concurrency

- C++11 provided `thread`, `async`
- Synchronization
 - `mutex`, `condition_variable`
 - `latch`, `barrier`, `semaphore`, `atomic_wait`
- Memory model
- Atomics



Time

- Current time
 - chrono
- Durations
 - Sleep
- No Timers
 - missing event mechanism



Networking

- Networking TS based on ASIO
 - event model problematic for some domains
 - will probably not be part of ISO C++
- New proposal expected
 - based on new event model
 - probably not in C++26



More C++ Support

- Minimal event support
 - interrupts are signals
- Event model part of Networking TS
 - based on callbacks
 - problematic lifetime issues
- Coroutines
 - still no real library support

- Filesystem



Standard C++ Library

- C++ provides much for small systems beyond synchronization and tasks
 - not only freestanding!



Standard Library

```
#include <string>
#include <vector>
#include <memory_resource>
#include <charconv>

constexpr size_t pmrSize = 1024;
std::byte pmrBuf[pmrSize];
StaticAllocator alloc(pmrBuf, pmrSize);
std::pmr::vector<int> vi({10, -24, 3456}, &alloc);
std::pmr::vector<std::pmr::string> vs(&alloc);
void fillVect()
{
    vi.push_back(-713);
    vs.push_back(std::pmr::string(" Hello", &alloc));
    vs.push_back(std::pmr::string(" AVR", &alloc));
}
```



execution

- C++ Standard Proposal (P2300)
 - heterogenous computing resources
 - `std::thread` is not sufficient
 - event handling model
 - aimed for C++26



Execution Contexts

- `std::thread` is insufficient
- Execution contexts provide agents to run tasks
 - `std::async`, `std::thread`
 - thread pool
 - ISR
 - other core (GPU)
- Execution contexts not part of API
- Execution contexts provide schedulers
- Schedulers have a function `schedule()`



Event Interface

- Many (embedded) systems are event triggered
- Sender/Receiver provides the standard interface for this
 - not:
`async_event(ev, callback);`
 - but:
`ev | callback;`
- Some details still open
- Networking proposal expected
- Planned for C++26



Data Concentrator

- Based on sender/receiver

```
exec::sender auto data1 = isr1()           // interrupt context
  | bufferedTransfer(4, highPrioFilt1Task) // switch to application
  | then(dealWithValue1); // runs in application context
exec::sender auto data2 = isr2()
  | bufferedTransfer(4, filt2Task)
  | then(dealWithValue2);
exec::sender auto concentrate = exec::when_any(data1, data2)
  | exec::then(validate)
  | exec::then(store);
forEver(concentrate);
```

- Concentrator using sender/receiver
 - two (ISR) producers, two separate filters
 - one common filter, one consumer
 - one producer has higher priority



Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



C++OS

- Implement full C++ on small systems
- Some extensions specific for small systems
- Native ARM
 - Cortex-M0, Cortex-M3/4
- Native 8-bit AVR
 - AVR6
- Hosted implementation on FreeRTOS
 - ESP32



Standard Library

- Lot of C++ Standard Library is OS independent
 - "freestanding"
- Even non-freestanding
 - `<chrono>`
 - `just now()` is missing
 - `<mutex>`
 - `lock_guard`, `unique_lock`
 - just `mutex` itself is missing
 - etc...
- GCC's `libstdc++` can be built e.g. for ARM, AVR, MSP430
 - C++OS simply fills the gaps



Queues

- Important communication mechanism
- Between different tasks
- Between ISRs and tasks
 - needs to be lock-free on ISR side



Sender/Receiver Queues

- The role of queues in the sender/receiver model is still largely open
- But the most important point of sender/receiver is customization – but the customization mechanism is still open
- Any queues between two stages can be customized to the involved execution agents



Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



Examples

- Simple Test example
 - setup some tasks and start them
 - nice test for correct porting



Examples

- Interrupts
 - transfer data from ISR to application

IRQ Context

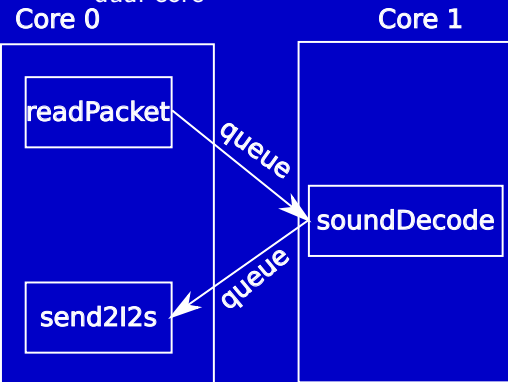
APP Context





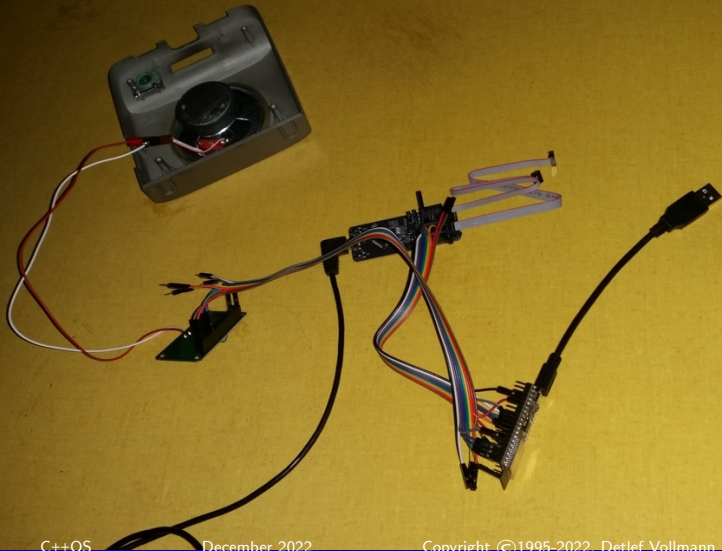
Examples

- Wireless speaker
 - receive sound stream from WiFi
 - decode it
 - send to I2S
 - dual core





Hardware





Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



Ports

- Not much to provide:

```
std::byte *estack;  
void volatile *make_stack(void volatile *stack_top,  
                           void *return_addr,  
                           void *function_addr,  
                           void *parameter);  
void yield();  
[[noreturn]] void initial_switch();  
uint8_t sys_tick(); // timer_tick and context switch  
  
struct interrupt_lock;  
void disable_interrupts();  
void enable_interrupts();  
bool interrupts_enabled();  
  
std::chrono::steady_clock::now() noexcept
```



Ports

- Initial port for ARMv7, Cortex-M4 w/o FPU
- Current primary port for AVR6
- Now also for ARMv6, Cortex-M0
- Non-native port for ESP32 Xtensa
 - based on ESP-IDF, FreeRTOS



sys_traits

- Definitions for your application

```
struct sys_traits
{
    static constexpr int static_task_count = 5;
    static constexpr int dynamic_task_count = 2;
    static constexpr int default_stack_size = 270;
    static constexpr int default_stack_guard_size = 4;
    static constexpr uint32_t default_stack_guard_value = 0xdead66aa;
    static constexpr int default_priority = 10;
    static constexpr int max_function_size = 8;
    static constexpr int min_stack_size = 160;
    static constexpr int main_stack_size = 180;
    static constexpr uint8_t max_event_count = 16;
};
```



Implementation

- Pretty simple
- Simple priority based round-robin scheduler
- event provides for blocking
- Library generally based on `#include_next`
- Using `disable_interrupt()` quite a lot
- No SMP



Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



Don't Use C++OS

- C++OS is a proof of concept
- C++OS is a toy to try out new C++ proposals
- Not for production use
 - buggy
 - no tests
 - incomplete
 - no interrupt policy
- C++OS provides no SMP support



OS Overhead

- RAM
 - full stack for each task
- Context switch overhead
 - store/restore registers and more
- Scheduling overhead
 - and not interruptible in C++OS



Overview

What and Why?

C++

C++OS

Examples

C++OS Implementation

Why C++OS?

References



Code

- Any code shown here can be found at
<https://gitlab.com/cppos1/cppos>
<https://gitlab.com/cppos1/sounddemo>



References

- These slides:
<http://www.vollmann.ch/>
- Implementation:
<https://gitlab.com/cppos1/cppos>
- Standard proposals, TSes:
<http://www.open-std.org/JTC1/SC22/WG21/docs/papers/>
- Executors:
[2022/p2300r4.html](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2022/p2300r4.html)
- Queues:
[2019/p0260r3.html](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2019/p0260r3.html)



Questions

- ???