kaspersky

Bare metal C++

Karina.Dorozhkina@kaspersky.com

Embedded == C?

```
if(buf[0] == 0x31 && insize == 8)
{
    CH_InMsg = 1;
}
else if(buf[0] == 0x33 && insize == 16)
{
    CH_InMsg = 1;
}
else if(buf[0] == 0x34 && insize == 16)
{
    CH_InMsg = 1;
}
else if(buf[0] == 0x35 && buf[1] == 0x11 && insize == 4)
{
    CH_InMsg = 1;
}
else if(buf[0] == 0x35 && buf[1] == 0x13 && insize == 12)
{
    CH_InMsg = 1;
}
```

Embedded == C?

```
#define DECLARE_PROXY_FOR_INTERFACE(iface) \
typedef struct iface##Proxy_\
{\
        iface##Vtbl vtbl;\
        struct iface##_proxy rawProxy;\
} iface##Proxy
```

Преимущества C++

- Шаблоны
- Полиморфизм
- STL и boost
- constexpr

- Придумай свой поинт

Наступление C++ на embedded

MISRA стандарт

- Рекомендации по написанию C и C++ кода для критических систем

(средства передвижения, медицинские системы)

- MISRA C создан в 1998 и является одним из главных стандартов разработки для критических систем
- Стандарты для C++ представлены слабее

MISRA C++ 2008
AUTOSAR расширение для C++14

- Статические анализаторы для MISRA (CodeSonar от GrammaTech, PVS Studio)

Примеры рекомендаций MISRA

- Управление динамической памятью

Rule 18–4–1 (Required) Dynamic heap memory allocation shall not be used.

- Не рекомендуется использовать любой код, потенциально приводящий к implementation-defined или undefined behavior

Rule 27–0–1 (Required) The stream input/output library <cstdio> shall not be used

- Разрешается использовать исключения с некоторыми ограничениями

Rule 15–5–1 (Required) A class destructor shall not exit with an exception.

C++ стандарт Freestanding proposal

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0829r4.html

Цель

- Выделить подмножество стандартной библиотеки, не требующей наличия ОС

Суть предложения

- Перечислить и разметить тегэми заголовочные файлы или часть их классов, доступных во freestanding среде
- Использование прочих заголовочных файлов во freestanding среде должно приводить к ошибке или warning

C++ стандарт Freestanding proposal

Не допускается ко включению во freestanding функционал, использующий:

- Динамическую память
- Операции с плавающей точкой
- Исключения
- RTTI

Примеры freestanding функционала

- Полностью включены <csetjmp> <utility> <tuple> <ratio> <compare> <contract> <ranges>
- Часть <memory>
  unique_ptr в роли RAII объекта
  make_unique использует heap и не включен во freestanding

- Большая часть <string_view>

Исключая функции basic_string_view::at, copy, substr, compare

- <array> исключая array::at
- Небольшое подмножество <cmath>

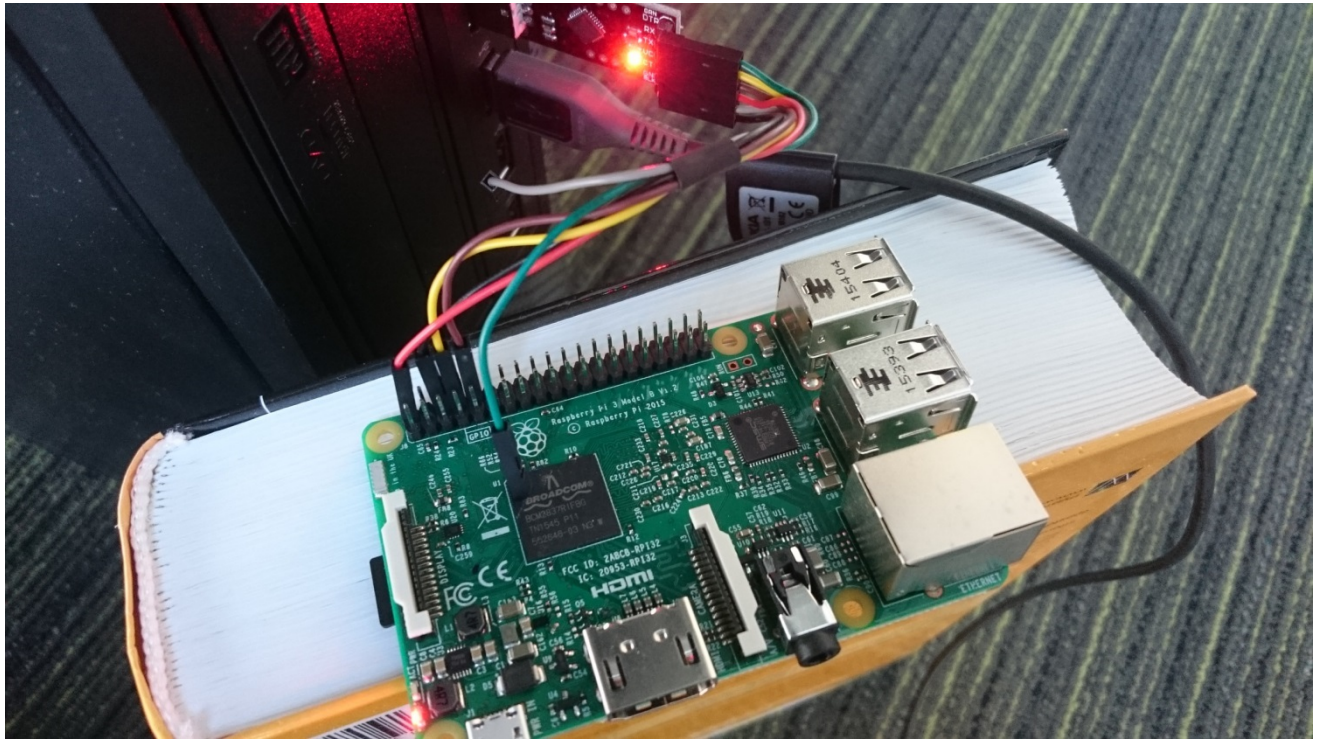
abs(int)
abs(long int)
abs(long long int)

А как же мои любимые контейнеры?

Выбор платформы для bare metal экспериментов

Raspberry Pi 3 Model B
Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Архитектура **ARMv8 Cortex-A53**

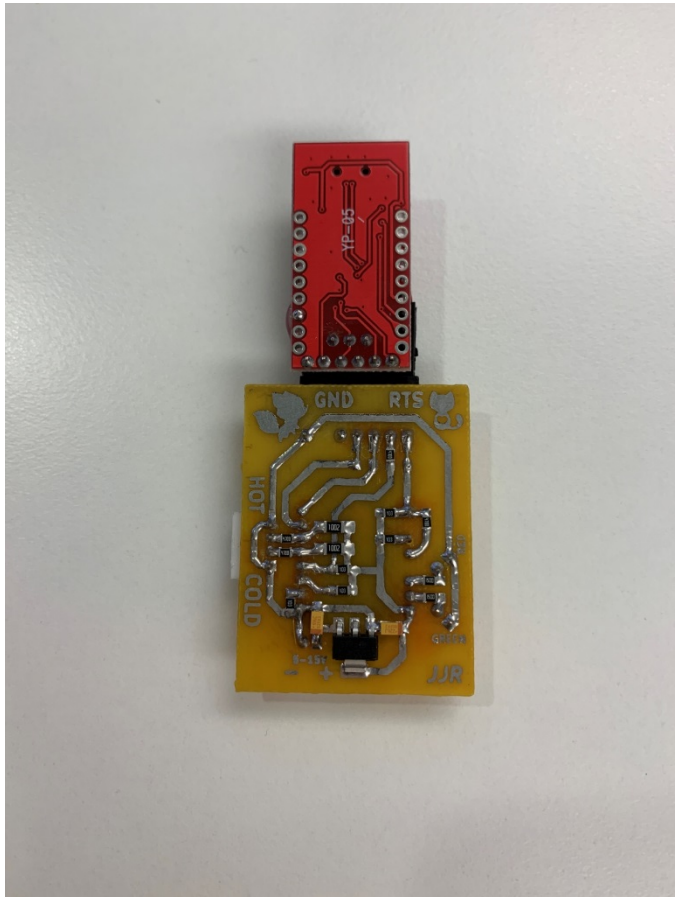
Тулчейн для сборки:
[GNU Toolchain for the A-profile Architecture]
AArch64 bare-metal target (aarch64-elf)
gcc 8.3

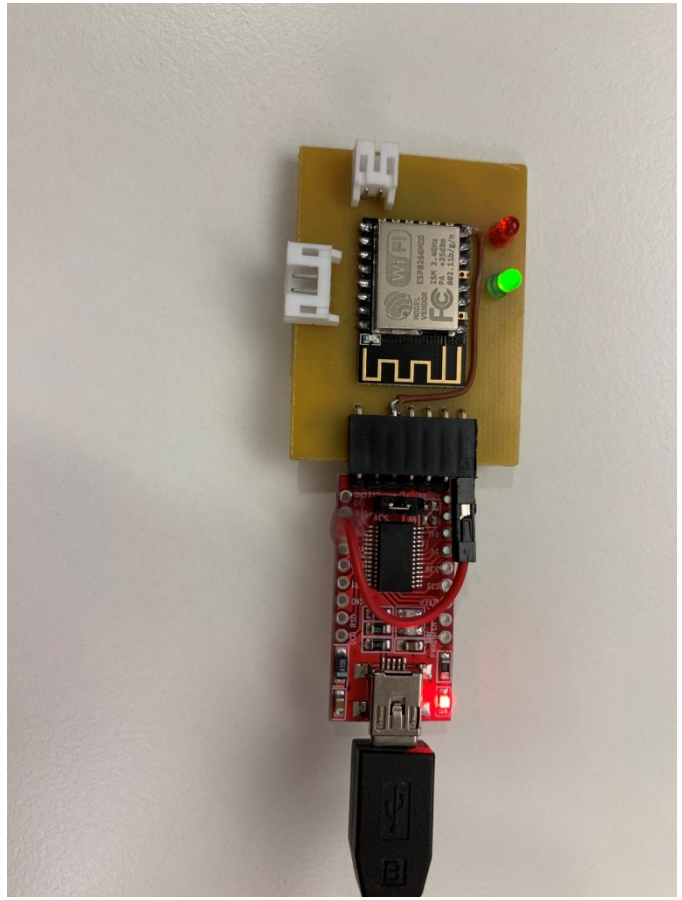


Если raspberry – не круто

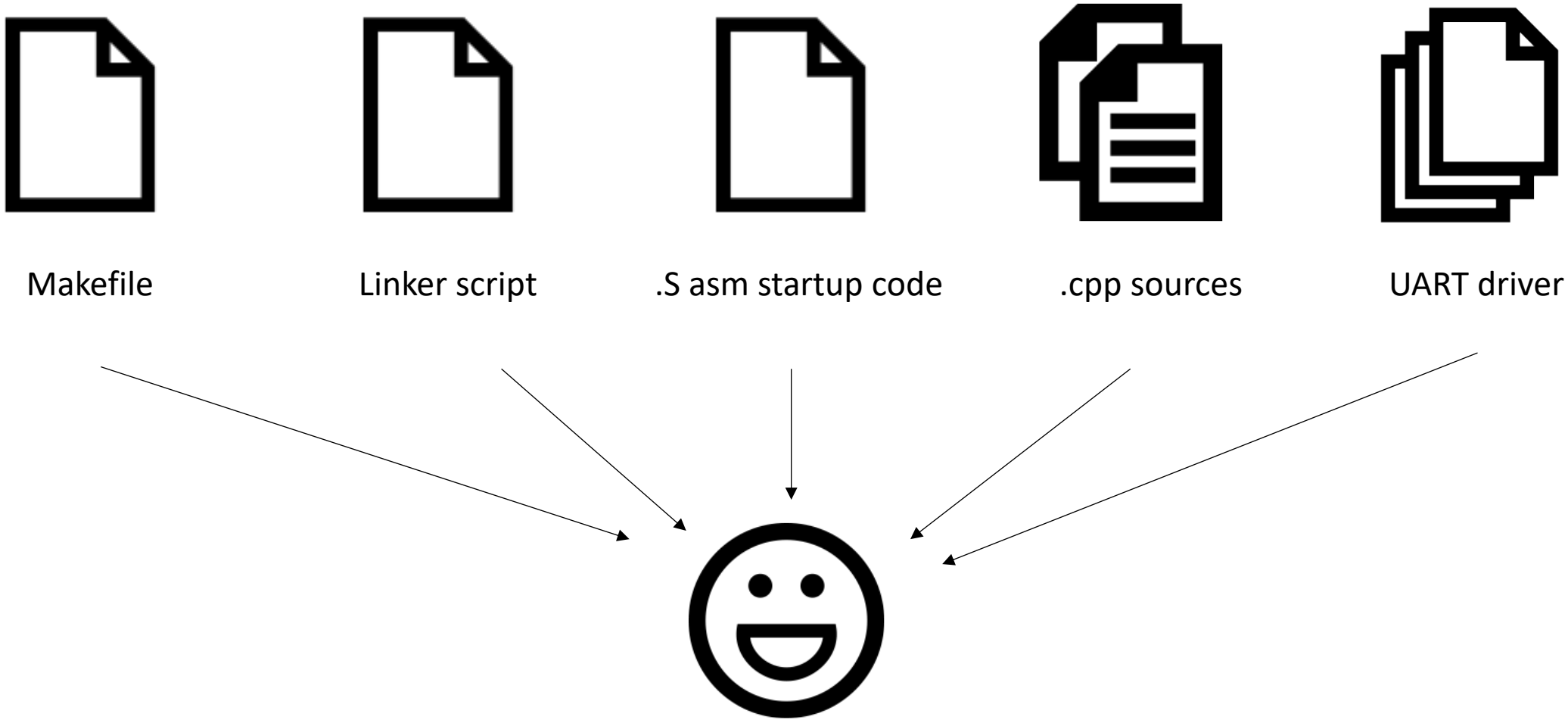
ESP8266
80 MHz 32-bit процессор **Tensilica Xtensa L106**
512Kb RAM

[Тулчейн для сборки]
gcc 4.8.2





Сборка bare metal решения



Makefile

Исключаем стандартные библиотеки линкера и startup код
-nostdlib

Не используем исключения
-fno-exceptions

Не используем rtti
-fno-rtti

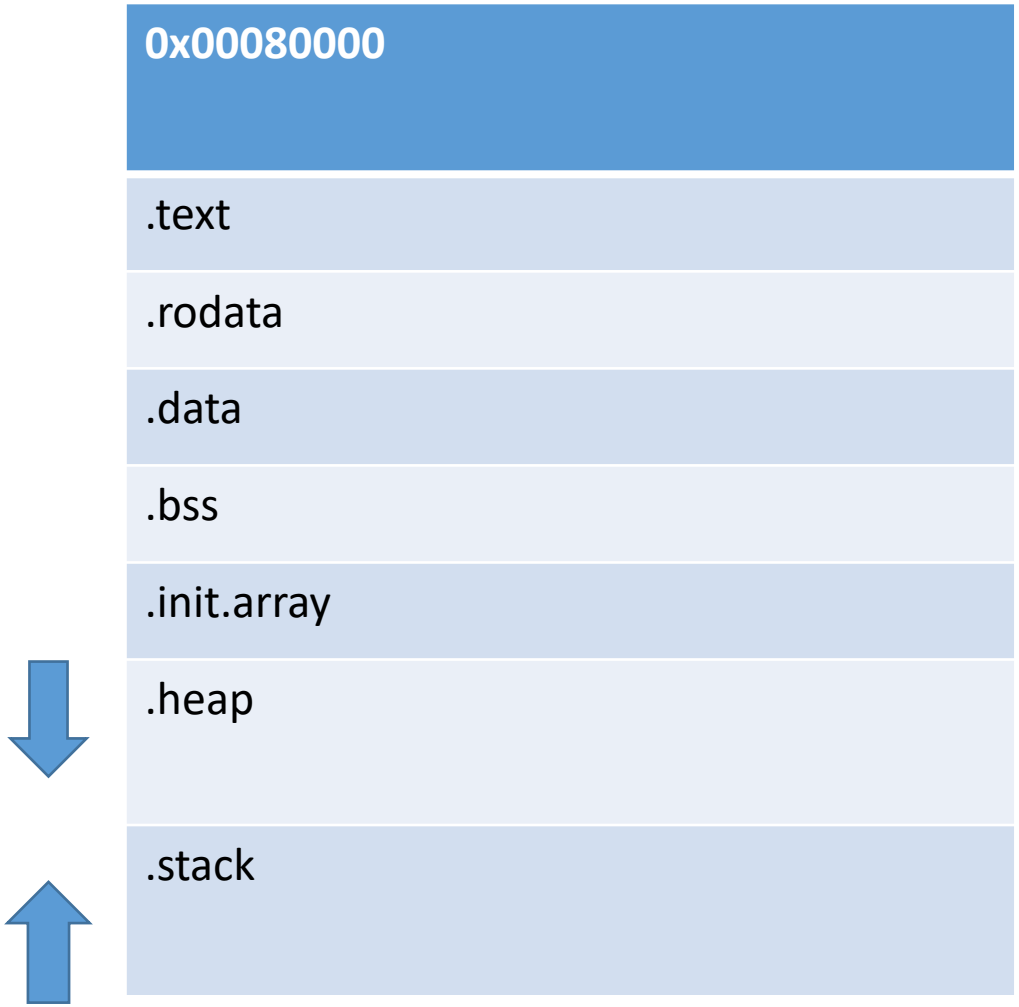
Указание своего скрипта для linker

```
kernel8.img: start.o $(OBJS)
	$(COMPILER_PATH)aarch64-elf-ld -nostdlib -nostartfiles start.o $(OBJS) -T load.ld -o kernel8.elf
```

Linker script

Определяет расположение секций в памяти
Адрес загрузки кода Raspberry 0x00080000

```
heap_size = 1M;
stack_size = 1M;

ENTRY(interrupt_vector)
SECTIONS
{
    . = 0x00080000;
    .text : { … }
    .rodata : { …}
    .data : {…}
    .bss : {
        __bss_start = .;
        *(.bss .bss.*)
        *(COMMON)
        __bss_end = .;
    }
    .init.array : {…}
    .heap : {…}
    .stack: {…}
    _end = .;
}
```



asm startup code

- Таблица векторов прерываний (адрес 0x00000000)
- Установка указателя на стек
- Зануление .bss секции
- Вызов конструкторов глобальных объектов
- Вызов main функции

```
interrupt_vector:
//interrupt vector handlers

start:
ldr x0, =stack_start
mov sp, x0

// clear bss
ldr     x1, =__bss_start
ldr     w2, =__bss_size
bss_loop :
cbz     w2, init_globals
str     xzr, [x1], #8
sub     w2, w2, #1
b       bss_loop

init_globals :
bl GlobalInitialize

bl main
b  start
```

Компилируем std::string на bare metal

```
#include <string>
void main()
{
    std::string s(“Hello, sick sad world!");
}
```

main.cpp: undefined reference to `operator delete(…)‘
main.cpp: undefined reference to `operator new(…)

Интерфейс std::basic_string

```
template<
        class CharT,
        class Traits = std::char_traits<CharT>,
        class Allocator = std::allocator<CharT>
>class basic_string;
```

since C++17

```
namespace pmr {
    template <class CharT, class Traits = std::char_traits<CharT>>
    using basic_string = std::basic_string< CharT, Traits,
                                            std::polymorphic_allocator<CharT>>
}
```

Краткая эволюция аллокаторов

C++03

template<typename T> class allocator :
- 7 typedefs
- 1 вложенный шаблон
- 2 конструктора
- 7 функций
- 2 оператора

C++11

template<typename T> class allocator :
- 1 typedef
- 2 конструктора
- 2 функции
- 2 оператора

C++03 аллокатор

stateless аллокаторы

All instances of a given allocator type are required to be interchangeable and always compare equal to each other.

```
template <typename T>
class my_allocator
{   …
    Pool pool;
};

using my_string = std::basic_string<char, std::char_traits<char>, my_allocator<char> >;

my_allocator<char> a1(pool1);
my_allocator<char> a2(pool2);

my_string s1("cpp", a1);
my_string s2("conf", a2);

s1.swap(s2); //??????????????????
```

C++11 аллокатор

statefull аллокаторы

```
template <typename T>
class my_allocator
{
    ...
    using propagate_on_container_copy_assignment = std::false_type | srd::true_type;
    using propagate_on_container_move_assignment = std::false_type | srd::true_type;
    using propagate_on_container_swap            = std::false_type | srd::true_type;
};
```

C++03 аллокатор

- **Нет поддержки fancy pointers**

boost::offset_ptr

```
template <class T>
class allocator
{
public:
        typedef T                 value_type;
        typedef T&                reference;
        typedef T const&          const_reference;
        typedef T*                pointer;
        typedef T const*          const_pointer;
        ...
};
```

C++11 аллокатор

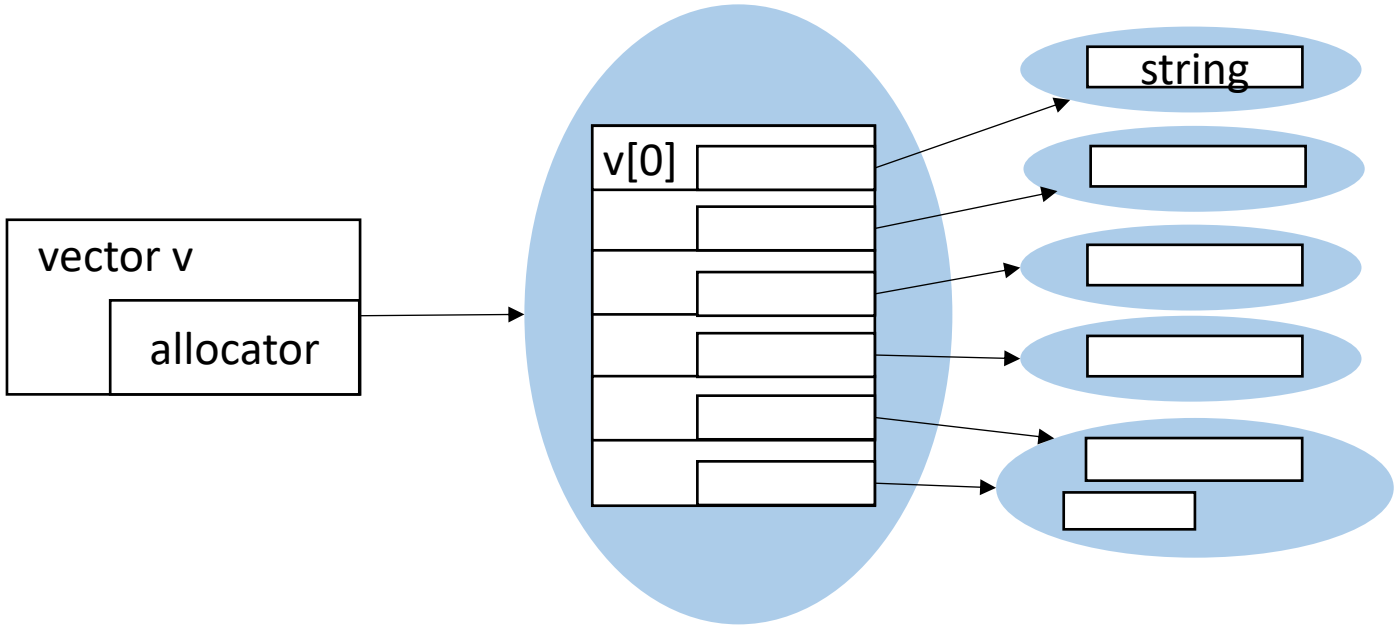
- **Поддерживает fancy pointers**

```
template< class Ptr > struct pointer_traits;
template< class T > struct pointer_traits<T*>;

template<class T> struct pointer_traits<T*>
{
        using pointer         = T*;
        using element_type    = T;
        using difference_type = ptrdiff_t;

        template<class U> using rebind = U*;

        static constexpr pointer pointer_to(see below r) noexcept;
};
```

C++03 аллокатор

- **Нет поддержки модели scoped allocator**

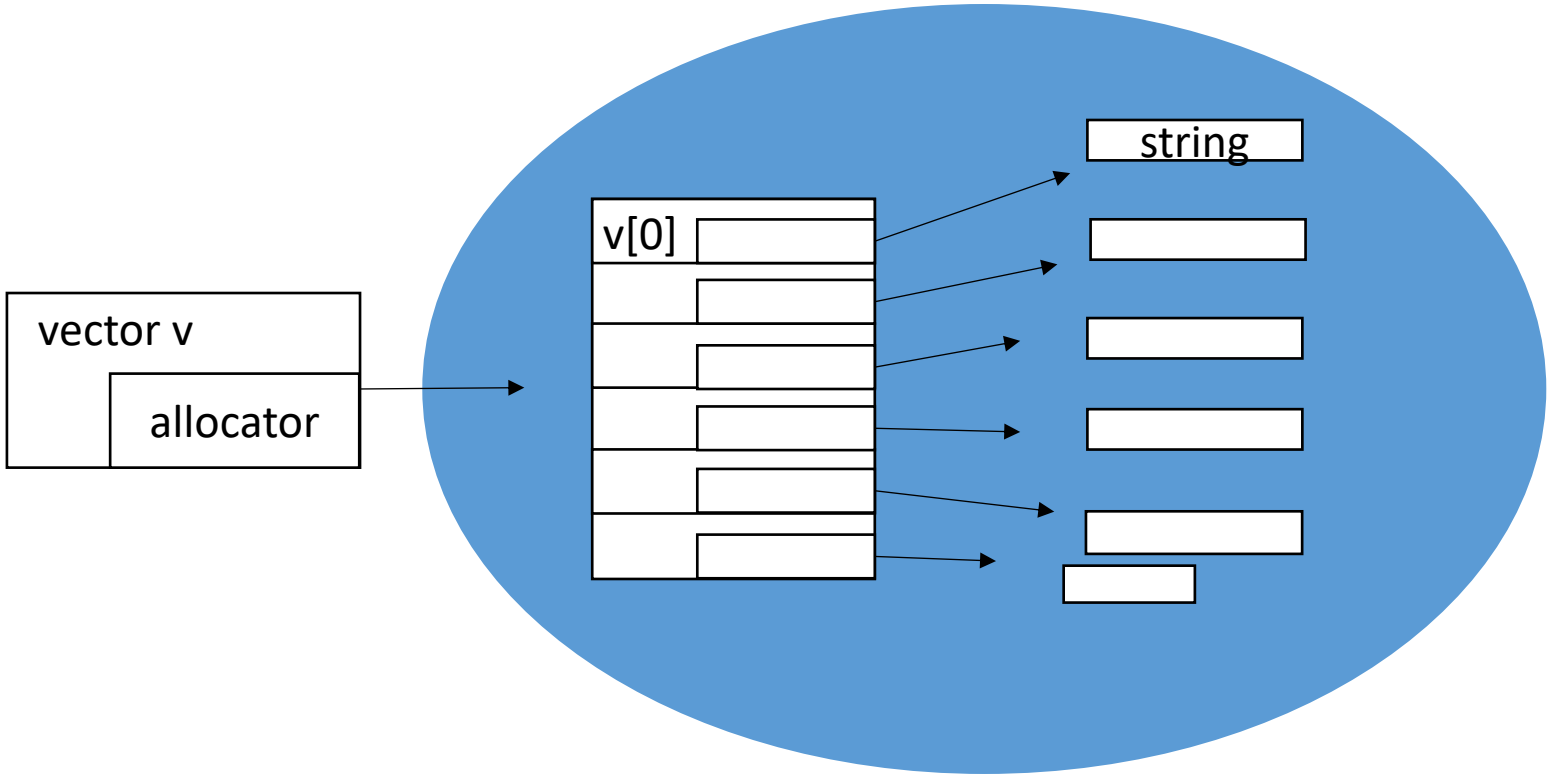
`std::vector<std::string>`



C++11 аллокатор

- **std::scoped_allocator_adapter**

```
namespace bi = boost::interprocess;
template<class T> using alloc = bi::adaptive_pool<T, bi::managed_shared_memory::segment_manager>;
using ipc_string = std::basic_string<char, std::char_traits<char>, alloc<char>>;
using ipc_vector = std::vector<ipc_string, std::scoped_allocator_adaptor<alloc<ipc_string>>>;
```



C++03 аллокатор

```
template <class T> class allocator
{
public:
                allocator() throw();
                allocator(const allocator&) throw();
                template <class U> allocator(const allocator<U>&) throw();
                ~allocator() throw();
                typedef allocator-specific pointer;
                typedef allocator-specific const_pointer;
                typedef allocator-specific reference;
                typedef allocator-specific const_reference;
                typedef allocator-specific value_type;
                typedef allocator-specific size_type;
                typedef allocator-specific difference_type;
                pointer address(reference x) const;
                const_pointer address(const_reference x) const;
                pointer allocate(size_type, allocator<void>::const_pointer hint = 0);
                void deallocate(pointer p, size_type n);
                void construct(pointer p, const T& val);
                void destroy(pointer p);
                size_type max_size() const throw();
                template <class U> struct rebind
                { typedef allocator<U> other; };
};
```

Минимальная имплементация аллокатора C++11

```
template <class T>
class minimal_allocator
{
public:
        using value_type = T;
        minimal_allocator() noexcept;
        template <class U> minimal_allocator(allocator<U> const&) noexcept;
        value_type* allocate(std::size_t n);
        void deallocate(value_type* p, std::size_t) noexcept;
};

template <class T, class U>
bool operator==(allocator<T> const&, allocator<U> const&) noexcept { return true; }

template <class T, class U>
bool operator!=(allocator<T> const& x, allocator<U> const& y) noexcept { return false; }
```

C++11 аллокатор

Интерфейс std::allocator_traits

```
void basic_string<CharT, Traits, Alloc>::destroy(size_type size) noexcept
{
    allocator_traits::deallocate(get_allocator(), data(), size + 1);
}
```

Интерфейс std::allocator_traits

```
template <class Alloc> struct allocator_traits
{
  using allocator_type            = Alloc;
  using value_type                = typename Alloc::value_type
  using pointer                   = Alloc::pointer or value_type*
  using const_pointer             = INFERRED
  using void_pointer              = INFERRED
  using const_void_pointer        = INFERRED
  using difference_type           = INFERRED
  using size_type                 = INFERRED

  using propagate_on_container_copy_assignment      = Alloc::propagate_on_container_copy_assignment
or std::false_type
  using propagate_on_container_move_assignment      = INFERRED
  using propagate_on_container_swap                         = INFERRED
  using is_always_equal(since C++17)                        = INFERRED
...
};
```

Интерфейс std::allocator_traits

```
template <class Alloc> struct allocator_traits
{
  …
  template <class T> using rebind_alloc = INFERRED
  template <class T> using rebind_traits = allocator_traits<rebind_alloc<T>>

  static pointer allocate(Alloc& a, size_type n); // until C++20
  static pointer allocate(Alloc& a, size_type n, const_void_pointer hint); // until C++20
  static void deallocate(Alloc& a, pointer p, size_type n);

  template< class T, class... Args >
  static void construct(Alloc& a, T* p, Args&&... args);

  template< class T >
  static void destroy(Alloc& a, T* p);

  static size_type max_size(const Alloc& a) noexcept;
  static Alloc select_on_container_copy_construction(const Alloc& a);
};
```

Реализация аллокаций в baremetal

Скрипт линкера

```
heap_size = 1M;

SECTIONS
{
    . = 0x80000;
    …
    . = ALIGN(16);
    .heap : {
        heap_start = .;
        . = . + heap_size;
        heap_end = .;
    }
    _end = .;
}
```

Способ управления памятью

- Размер heap’а 1 Мб
- Память аллоцируется равными блоками по 64 байта
- Free Bitmap - дополнительная структура данных для хранения состояния блока

1 – свободен

0 – занят

Блок 1	Блок 2				
0x00080E00	0x00080E40	…			0x00081200 FIX

0	1	1	1	1

Пример реализации BitMapHeap

```
extern unsigned char heap_start;
extern unsigned char heap_end;

class BitMapHeap
{
public:
    using pointer = void*;
    using size_type = std::size_t;

    static const size_type BlockSize = 64;
    static const size_type MaxHeapSize = 1048576;

    pointer allocate(size_type n);
    void deallocate(pointer p, size_type n);
private:
    std::array<bool, MaxHeapSize / BlockSize> m_bitMap;
…
}
```

Stateless аллокатор

```
static BitMapHeap heap;

template <class T>
class BaremetalAllocator
{
public:
  using value_type = T;

  …
  pointer allocate(size_type n, BaremetalAllocator<T>::const_pointer hint = 0)
  {
    return heap.allocate(n * sizeof(T));
  }

  void deallocate(pointer p, size_type n)
  {
    heap.deallocate(p, n * sizeof(T));
  }
};
```

Инициализация глобальных объектов

start.S

```
.extern GlobalInitialize
...
init_globals :
bl GlobalInitialize

bl main
```

Инициализация глобальных объектов

global_objects.cpp

```
extern void(*init_array_start[])(void);
extern void(*init_array_end[])(void);

extern "C"
void GlobalInitialize()
{
    size_t count = init_array_end - init_array_start;
    for (size_t i = 0; i < count; ++i)
    {
        init_array_start[i]();
    }
}
```

Деинициализация глобальных объектов

```
extern "C"
int __cxa_atexit(
            void *object,
            void(*destructor)(void *),
            void *dso_handle)
{
    static_cast<void>(object);
    static_cast<void>(destructor);
    static_cast<void>(dso_handle);
    return 0;
}

void* __dso_handle = nullptr;
```

Либо:
Использовать флаг **-fno-use-cxa-atexit**

Статические объекты

- Thread-safe инициализация локальных статических объектов C++11

```
main.cpp: undefined reference to `__cxa_guard_acquire'
main.cpp: undefined reference to `__cxa_guard_release‘
```

-fno-threadsafe-statics

Компилируем std::basic_string с кастомным аллокатором

```
#include <string>
#include “baremetal_allocator.h"


using baremetal_string = std::basic_string<char, std::char_traits<char>,
BaremetalAllocator<char> >;
void main()
{
    baremetal_string s(“Hello, sick sad world");
}
```

main.cpp: undefined reference to `memset'
main.cpp : undefined reference to `memcpy'

Standart C функции

```
extern "C"
void* memcpy(void* dst, const void* src, std::size_t
n)
{…}

extern "C"
void* memset(void* ptr, int value, std::size_t n)
{…}

extern "C"
std::size_t strlen(const char* str)
{…}

extern "C"
void *memchr(const void *arr, int c, size_t n)
{…}
```

Зависимость от исключений

main.cpp: undefined reference to `std::__throw_length_error(char const*)‘
main.cpp: undefined reference to `std::__throw_logic_error(char const*)'

GNU libstdc++

```
#if __cpp_exceptions
void __throw_length_error(void)
{
    throw length_error();
}
#else
void __throw_length_error (void)
{
    abort();
}
#endif
```

Зависимость от исключений

```
namespace std {
void __throw_length_error(char const*)
{
        while (true) {}
}

void __throw_logic_error(char const*)
{
        while (true) {}
}
}
```

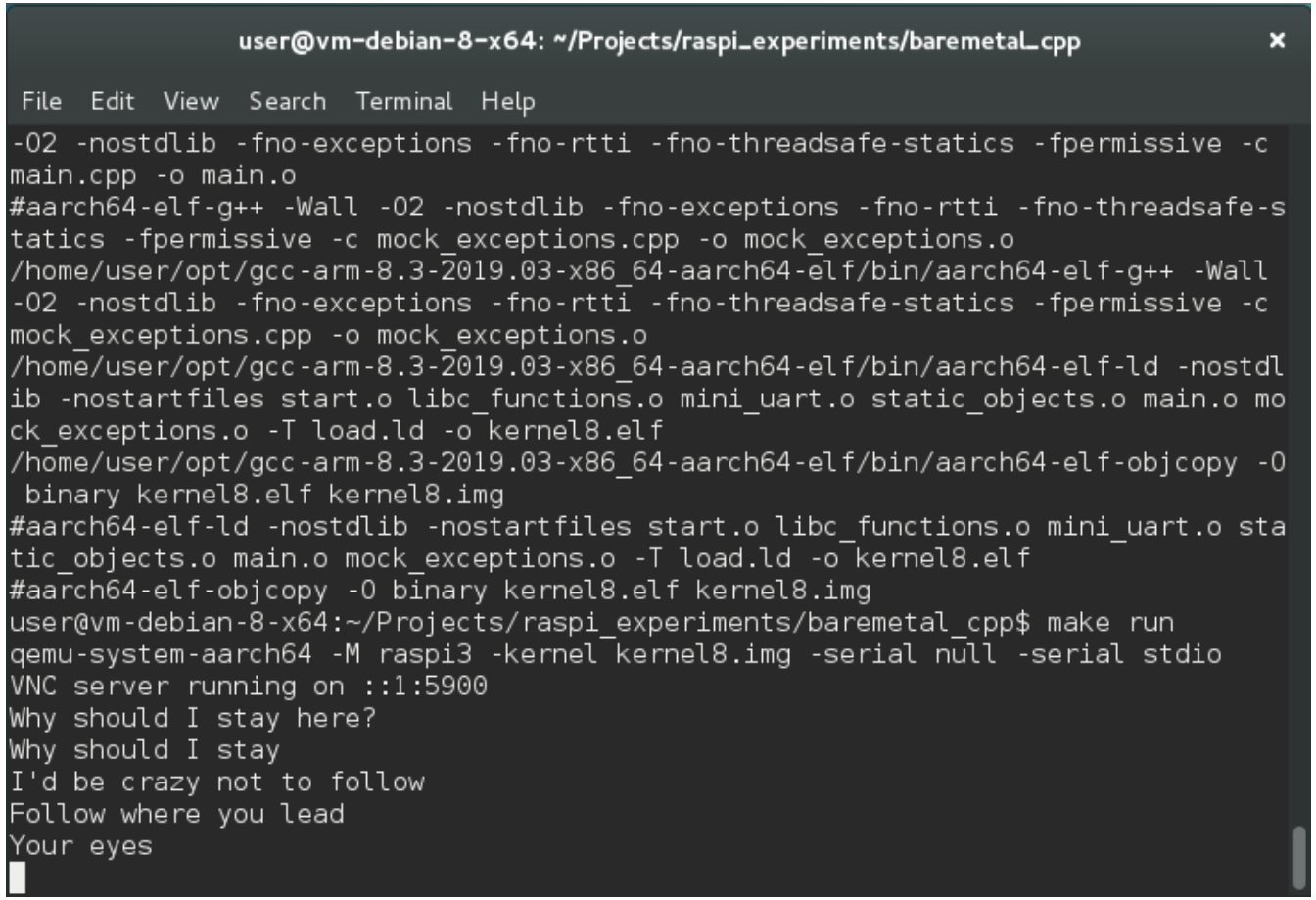
```
using baremetal_string = std::basic_string<char, std::char_traits<char>,
BaremetalAllocator<char> >;
using baremetal_vector = std::vector<baremetal_string, BaremetalAllocator<baremetal_string>
>;


void main()
{
    baremetal_string greeting("Why should I stay");
    greeting.append(" here?");
    cout << greeting.c_str() << "\n";

    std::size_t pos = greeting.rfind("h");
    baremetal_string part(greeting.begin(), greeting.begin() + pos);
    cout << part.c_str() << "\n";

    baremetal_vector phrase{ "I'd be crazy not to follow", "Follow where you lead" };
    phrase.push_back("Your eyes");
    for (const auto& word : phrase)
        cout << word.c_str() << "\n";
}
```

Подводим итоги

- Имплементация стратегии аллоцирования
- Имплементация аллокатора
- Инициализация глобальных объектов
- Реализация некоторых Standard C функций (memset/memcpy)
- Зависимость от исключений

Собери свой baremetal

https://github.com/keyrnk/baremetal_experiments



Дальнейшие эксперименты

- Убираем –fno-exceptions и реализовываем обработку исключений
- Реализовываем и сравниваем различные стратегии аллоцирования памяти
- Инициализируем и используем MMU

Полезные ссылки и ресурсы

Аллокаторы и стратегии аллоцирования

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2554.pdf

http://www.ecs.csun.edu/~cputnam/Comp%20322/lectures/Memory%20Allocation.pdf

https://accu.org/content/conf2013/Frank_Birbacher_Allocators.r210article.pdf

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3916.pdf

libstdc++ docs

https://gcc.gnu.org/onlinedocs/gcc-9.2.0/libstdc++/api/files.html

Написание linker script

https://www.eecs.umich.edu/courses/eecs373/readings/Linker.pdf

Embedded

http://caxapa.ru/thumbs/717646/effectcppemb.pdf

kaspersky

Let’s talk?

Karina.Dorozhkina@kaspersky.com