ЭФФЕКТИВНОЕ ИСПОЛЬЗОВАНИЕ ПАМЯТИ ВО ВСТРАИВАЕМЫХ СИСТЕМАХ

Алешкин Алексей

Старший разработчик отдела встраиваемых индустриальных решений

Обо мне

Алешкин Алексей

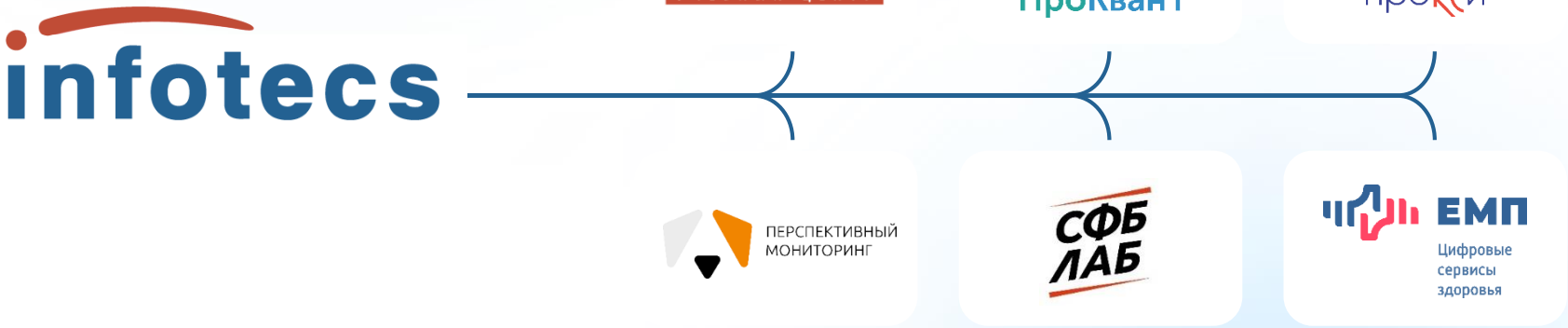
Старший разработчик отдела встраиваемых индустриальных решений

АО «ИнфоТеКС»

aleksey.aleshkin@infotecs.ru

Группа компаний «ИнфоТеКС»

2024



ИнфоТеКС в цифрах

Топ-10

Входит в десятку крупнейших ИТ-компаний в области разработки ПО

12 офисов

по всей стране

>10 млн

рабочих станций, защищенных продуктами ViPNet

>60 продуктов

для защиты информации

>30 лет

Работы на рынке ИБ

>1800

сотрудников

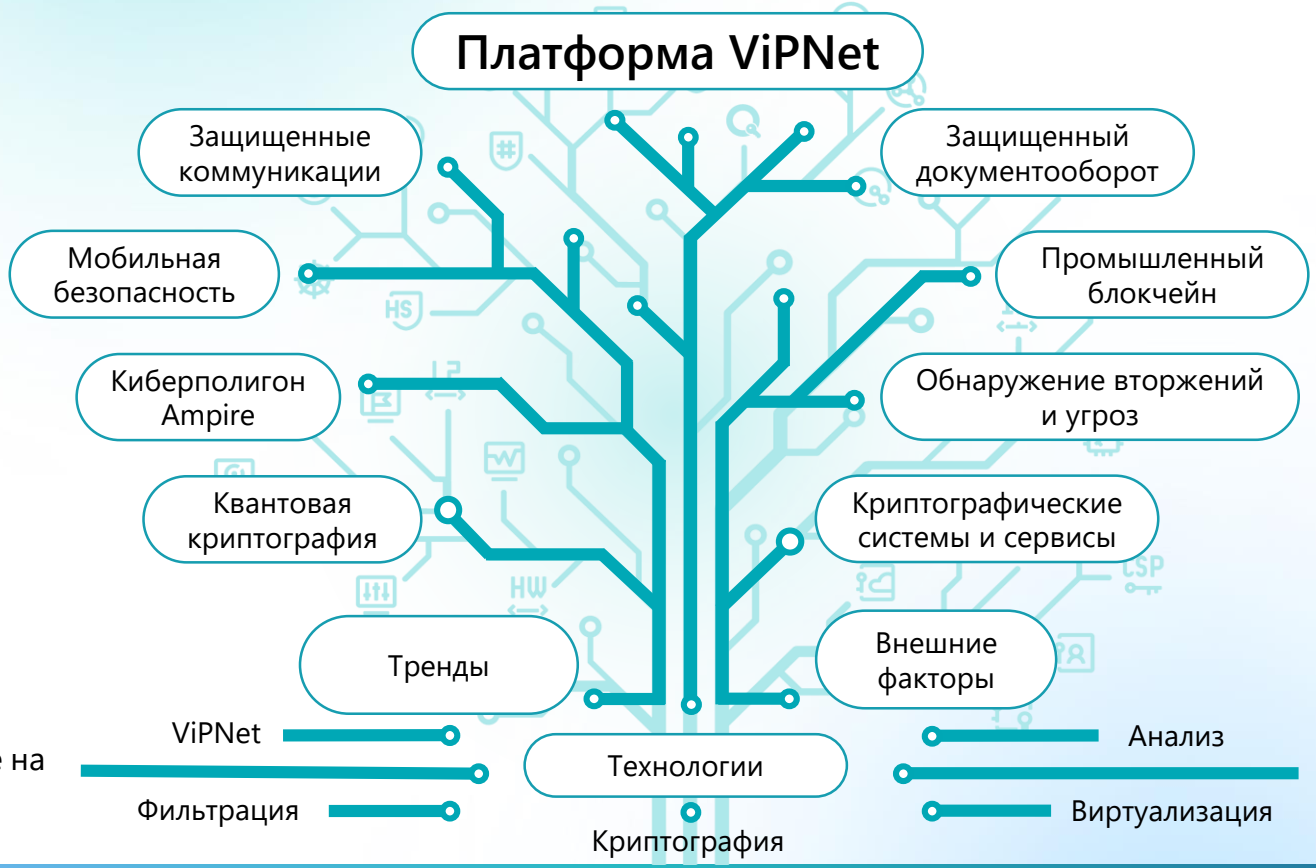
Топ-5

Входит в пятерку компаний по количеству патентов в области цифровых технологий

Представительства ИнфоТеКС

- Москва
- Санкт-Петербург
- Хабаровск
- Томск
- Уфа
- Ростов-на-Дону
- Екатеринбург
- Пенза
- Новосибирск
- Владивосток
- Красноярск
- Рязань

Продукты и решения ИнфоТеКС



План доклада

1. Особенности ViPNet SIES Core
2. Эффективное использование памяти Flash
3. Эффективное использование динамической памяти
4. Причины фрагментации динамической памяти
5. Способы снижения фрагментации на стадии разработки
6. Контексты аллокатора и их мультиплексирование

Особенности ViPNet SIES Core

Индустриальный криптографический модуль

Платформа Cortex-M4

Частота ядра: 160 – 220 МГц

Встроенная память

Опционально разнородная Flash для ПО от 2 до 3 Мб

Разнородная RAM для ПО и данных от 256 до 768 кб

Операционная система FreeRTOS

ОС реального времени

Вытесняющая многозадачность

Особенности ViPNet SIES Core

Индустриальный криптографический модуль

Интеграция с защищаемым устройством (ЗУ)

Особенности ViPNet SIES Core

Ключевые требования

Высокая доступность модуля для ЗУ

ЗУ должно знать, когда модуль окажется недоступным для обработки данных и на сколько времени

Потоковая обработка данных

Любая криптооперация должна иметь возможность проводиться с теоретически неограниченным по объему массивом данных

Стабильность производительности

Время исполнения криптоопераций должно оставаться в пределах 5% при идентичных условиях и настройках

Особенности ViPNet SIES Core

Основной язык разработки – C++ (стандарт 17)

Достоинства при разработке перед C

Парадигмы ООП, обобщенного и функционального программирования

Следование лучшим архитектурным паттернам и принципам

Повышение скорости разработки, рефакторинга и сопровождения

Возможность применения более широкого спектра библиотек

Особенности ViPNet SIES Core

Основной язык разработки – C++ (стандарт 17)

Недостатки при разработке встраиваемых устройств

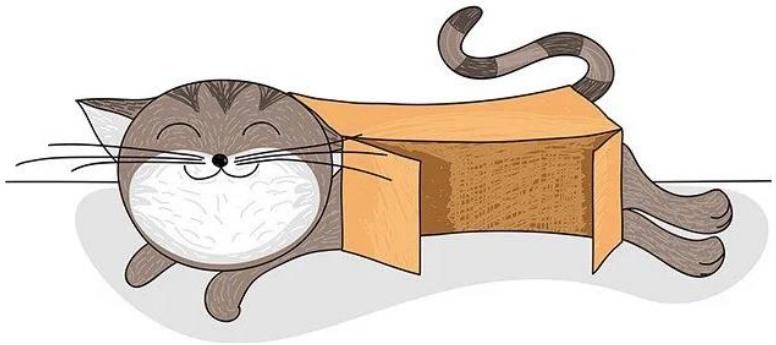
Объемные библиотеки и большое количество порождаемых типов при ограниченной памяти Flash

Неявное управление динамической памятью при ограниченной ОЗУ

Потенциальные потери производительности

Эффективное использование памяти Flash

Объем исполняемого файла становится больше размера Flash



Что увеличивает объем исполняемого файла:

- Поддержка новых коммуникационных интерфейсов
- Поддержка новых аппаратных платформ
- Поддержка новых криптографических алгоритмов
- Расширение бизнес-логики
- И все остальные пожелания продакт-менеджера

Эффективное использование памяти Flash

Карта решения

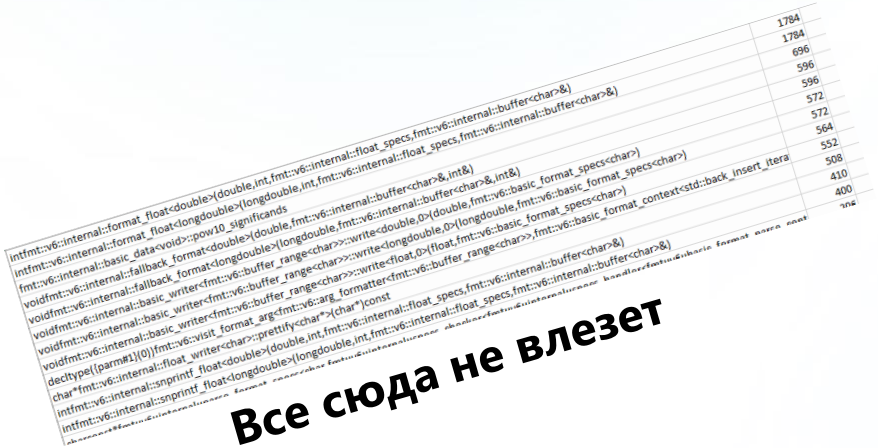
Подсчет размеров пространств имен в объектных файлах

Поиск альтернатив «тяжелым» библиотекам

Лайфхак: Уменьшение количества порождаемых типов

Эффективное использование памяти Flash

Что же заняло столько места?



Подсчет размеров пространств имен в объектных файлах

Пример выявления больших по размеру пространств имен в новой версии ПО

Вид изменения	Общий размер, байт
Изменено символов	-5506
Добавлено символов	268405
Удалено символов	60199
Итого	**202700**
Использование символов stream добавило	115360
Использование символов fmt добавило	31639
Использование символов locale добавило	37173
Изменение за счет stream, fmt и locale	**184172 (90,86%)**

Эффективное использование памяти Flash

Как от этого избавиться?

Поиск альтернатив «тяжелым» библиотекам и отказ от некоторых из них

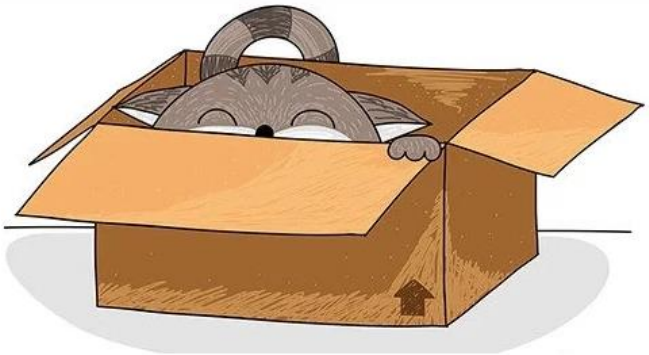
1. Отказ от boost, *stream и других библиотек, которые сразу занимают много памяти

2. Использование **ETL | itlib (MIT License)**
   - span
   - flat_map
   - static_vector
   - string_view
   - any
   - и т.д.

Все это привело к уменьшению бинарного файла более чем на 400кБ (20% от всего объема памяти Flash)

Эффективное использование памяти Flash

И что получилось в итоге?



Поиск альтернатив «тяжелым» библиотекам и отказ от некоторых из них

Привели к уменьшению бинарного файла более чем на 400кБ

Вид изменения	Общий размер, байт
Изменено символов	513276
Добавлено символов	345136
Удалено символов	-1289144
Итого	**-430732**

Эффективное использование памяти Flash

Обещанный лайфхак

Выиграно ~80 кБ
только на стандартных типах

Уменьшение количества порождаемых типов

Пример:

```
template<typename T>
constexpr T tuple_max(T max, T v)
{
    return max < v ? v : max;
}

template<typename T>
constexpr T tuple_max(T max, T v, unsigned int sizes...)
{
    return tuple_max(max < v ? v : max, sizes);
}

template<typename T1, unsigned int N, unsigned int maxSize>
constexpr auto generate_arrays ()
{
    return std::array<std::array<T1, maxSize>, N>{};
}

template<typename T1, unsigned int... sizes>
constexpr auto make_arrays_with_max_size()
{
    constexpr auto tpl_size = sizeof...(sizes);
    constexpr unsigned int max_size = tuple_max(0u, sizes...);
    return generate_arrays<T1, tpl_size, max_size>();
}

    auto [v1, v2, v3] = make_arrays_with_max_size<int, 3, 4, 5>();
```

Эффективное использование динамической памяти

После длительной работы под нагрузкой модуль «умирает»

Нагрузочное тестирование: циклически и непрерывно модуль выполняет разные криптооперации с разными объемами данных и на разных скоростях по всем коммуникационным интерфейсам

При работе модуля возникает исключение std::bad_alloc через дни непрерывной работы

Эффективное использование динамической памяти

Карта решения

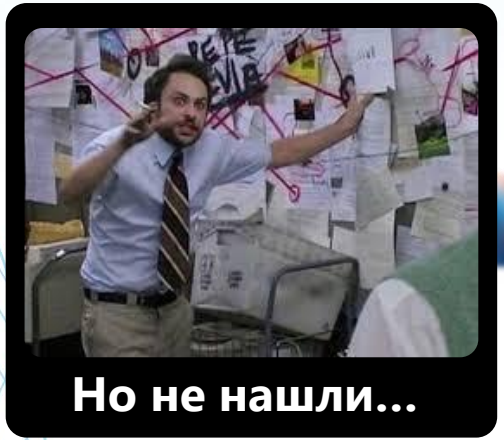
Сбор данных об использовании Heap

Анализ данных об использовании Heap

Минимизация фрагментации памяти на стадии разработки

Эффективное использование динамической памяти

Искали утечки



Сбор данных об использовании Heap

Структура mallinfo содержит важные данные, позволяющие оценить состояние heap в моменте

- arena – зарезервированная память
- uordblks – вся аллоцированная память
- fordblks – вся свободная память
- keepcost – свободная память на вершине

Эффективное использование динамической памяти

Зато нашли фрагментацию

Легенда:

- свободные участки
- аллоцированные до работы сценария участки
- аллоцированные в процессе работы сценария участки

Сбор данных об использовании Heap

keepcost < fordblks

В процессе keepcost уменьшается

И не возвращается полностью по окончании текущего сценария

Но будет изменен на следующем сценарии

keepcost < fordblks означает наличие фрагментации

Эффективное использование динамической памяти

infotecs

Микроконтроллер – это «черный ящик»

Сбор данных об использовании Heap

1. Использование функций __wrap_malloc, __wrap_free и ключей сборки gcc –-wrap=<symbol>
2. Использование ключа сборки gcc –finstrument-functions
3. Использование Semihosting FS для логирования данных на хостовую ФС
4. Зная адреса вызванных функций и выделяемый размер, можно сопоставлять вызовы выделений памяти с кодом

Эффективное использование динамической памяти

Что оказалось внутри...

Вот столько выделяется памяти на отрезках «кучи» при работе без кеширования данных в «куче»

Анализ данных об использовании Heap

После сценария освобождается вся память

Эффективное использование динамической памяти

Что оказалось внутри...

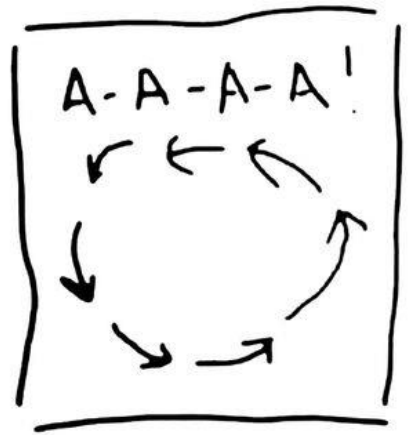
А столько выделяется памяти на отрезках «кучи» при работе с кешированием

Анализ данных об использовании Heap

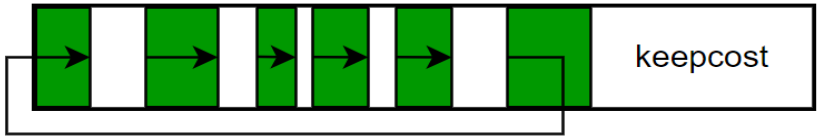
После сценария не освобождается кэш

Эффективное использование динамической памяти

Аллокатор newlib



Анализ данных об использовании Heap



- Оптимально по расходу памяти
- Проблемы с фрагментацией памяти
- Проблемы с производительностью

Причины фрагментации динамической памяти

Что же выяснилось в итоге?

Архитектор проекта, когда узнал, что >30% «кучи» фрагментировано

Минимизация фрагментации памяти на стадии разработки

Причины фрагментации:

1. Реаллокация
2. Множественная мелкая аллокация
3. Кеширование данных и объекты с долгим временем жизни
4. Отложенное освобождение
5. Особенности реализации стандартного аллокатора

Способы снижения фрагментации на стадии разработки

infotecs

Дело в векторах, но не только...

Минимизация фрагментации памяти на стадии разработки

Реаллокация при использовании std::vector.

Практики:

1. Предварительное резервирование памяти функцией reserve ровно под задачу
2. Отказ от shrink_to_fit
3. Использование static_vector и array
4. Замена на std::deque

Способы снижения фрагментации на стадии разработки

И в других контейнерах, но не только…

Минимизация фрагментации памяти на стадии разработки

Наличие мелких аллокаций и больших объемов метаинформации при использовании std::map и std::unordered_map

Практики:

1. Замена на flat_map везде, где возможно, исходя из объема данных
2. Использование array<pair<T1, T2>, N>, по возможности

Контексты аллокатора и их мультиплексирование

А может можно локализовать фрагментацию?

Переключение контекстов стандартного аллокатора согласно текущему сценарию

1. Реализация поддержки назначаемого контекста в стандартном аллокаторе
2. Инициализация отдельного контекста для каждого модуля или сценария, реализующего кеширование или отложенное освобождение
3. Реализация шаблона обертки, переключающей контекст аллокатора для каждого модуля или сценария

Контексты аллокатора и их мультиплексирование

Вариант 1

Мультиплексирование контекстов в коде с помощью обертки или с помощью --wrap

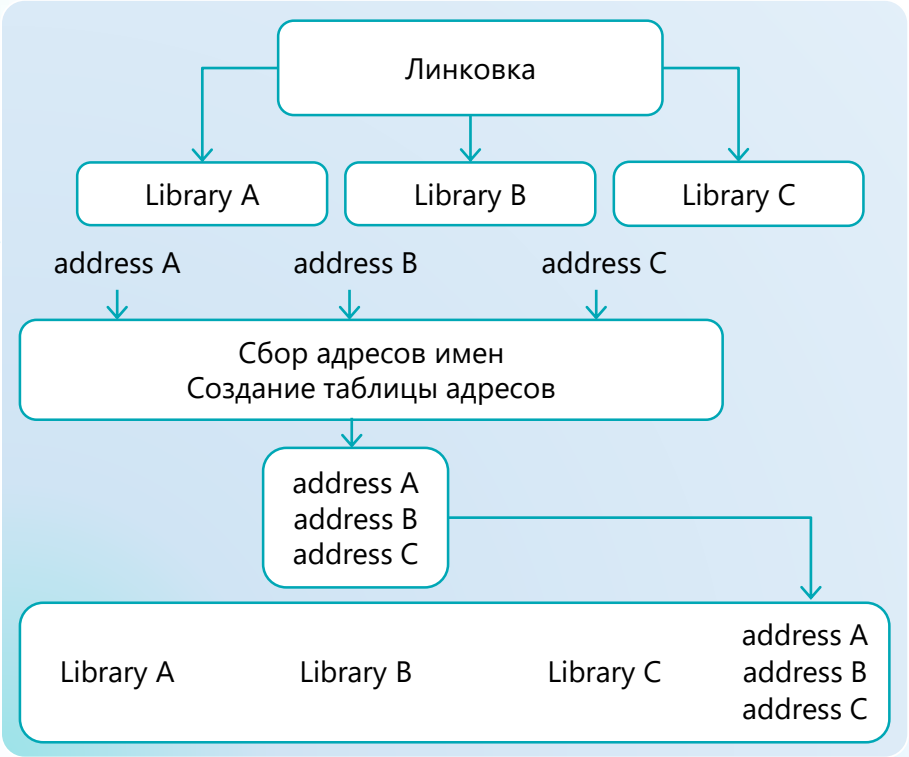
Переключение контекстов стандартного аллокатора согласно текущему сценарию

```
// Инициализация
struct AllocCtx {
    AllocCtx() = default;
    AllocCtx(size_t start_addr, size_t size){}
};

template<typename WrapperType>
AllocCtx init_alloc_ctx() {
    size_t start_addr = ::get_free_heap_address();
    return AllocCtx(start_addr, WrapperType::get_alloc_size());
}

// Использование
class ThirdPartyLibWrapper {
    AllocCtx alloc_ctx;
    constexpr static size_t heap_size = HEAP_VOLUME;

    ThirdPartyObject obj;

public:
    constexpr static size_t get_alloc_size() { return heap_size; }

    ThirdPartyLibWrapper(): alloc_ctx(init_alloc_ctx<ThirdPartyLibWrapper>())
    {}

    void ThirdPartyFn() { ::set_alloc_ctx(&alloc_ctx); obj.Fn(); }
};
```

Контексты аллокатора и их мультиплексирование

Вариант 2

Мультиплексирование контекстов с помощью линковки



Контексты аллокатора и их мультиплексирование

Вариант 2

Мультиплексирование контекстов с помощью линковки

```
// Инициализация
extern size_t* lib_addresses;
std::vector<size_t, AllocCtx> alloc_ctx;

struct AllocCtx {
    AllocCtx() = default;
    AllocCtx(size_t start_addr, size_t size): heap_addr(start_addr),
alloc_size(size) {}
    size_t heap_addr = 0;
    size_t alloc_size = 0;
};

void init_alloc_ctx(size_t heap_size) {
    while(*lib_addresses) {
        alloc_ctx.emplace_back(*lib_addresses, heap_size);
        ++lib_addresses;
    }
}

void __cyg_profile_func_enter(void* this_fn, [[maybe_unused]] void* call_site) {
    auto it = std::find_if(alloc_ctx.begin(), alloc_ctx.end(),
        [this_fn](const AllocCtx& ctx) {
           return ctx.heap_addr <= this_fn && this_fn < (ctx.heap_addr +
ctx.alloc_size);
        });
    if (it != alloc_ctx.end()) {
        ::set_alloc_ctx(*it);
    }
}
```

Ответы на вопросы

infotecs

infotecs.ru

Карьера

infotecs.team