

Кроссплатформенная разработка. Опыт Opera Software.

Алексей Хлебников

LVEE 2015

Зачем это нужно

- ▶ Больше платформ - больше пользователей
- ▶ Одна программа для всех платформ - меньше затраты на разработку
- ▶ Иммунитет к vendor lock-in
- ▶ Трудно или невозможно ошибиться выбором платформы
- ▶ Легче добавлять поддержку новых платформ
- ▶ Приложение, как правило, будет спроектировано лучше

Платформы, которые поддерживал браузер Opera

- ▶ Desktop, в том числе FreeBSD, Solaris, OS/2, BeOS и QNX
- ▶ Планшеты
- ▶ Smartphone, в том числе Symbian/Series60, Maemo/Meego, Bada, BlackBerry, Windows CE/Mobile
- ▶ Feature phone, в том числе BREW, P2K, EPOC, Psion, Java ME
- ▶ Игровые консоли, в том числе Playstation 3, Nintendo Wii и Nintendo DS
- ▶ Smart и не-smart TV, TV-приставки
- ▶ Автомобильные компьютеры
- ▶ Панель управления башенного крана

Возникающие проблемы

- ▶ Разные скорости CPU и объёмы памяти
- ▶ Разные размеры экранов и их количество
- ▶ Разные устройства управления
- ▶ Разные дополнительные устройства (камера, GPS, etc)
- ▶ Разные компиляторы, в том числе и капризные (ADS)
- ▶ Особые методы запуска приложений (BREW, P2K)
- ▶ Разные API у разных операционных систем

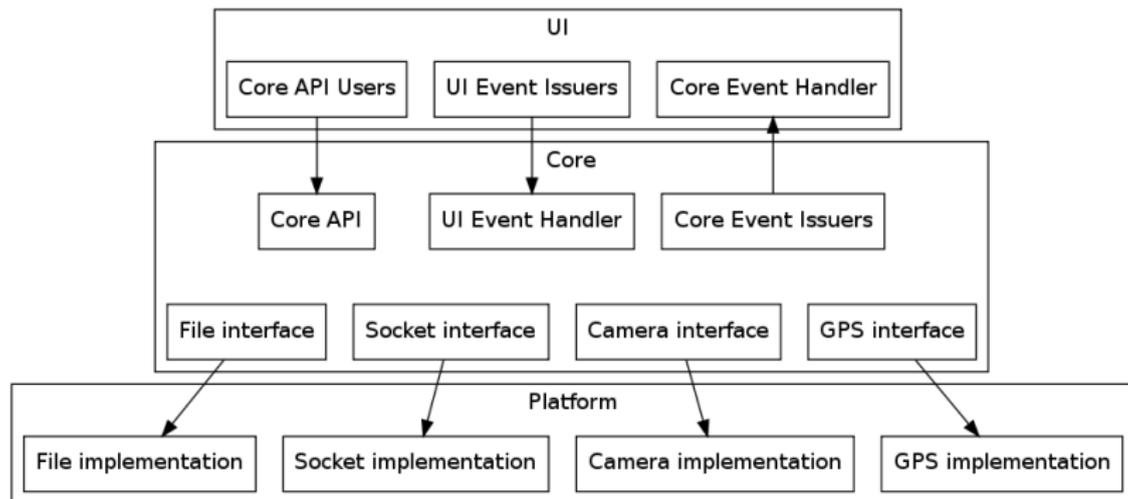
Требования Opera 8.5x/Presto 1.0 (2005~2011)

- ▶ 32-битный процессор
- ▶ 8 МБ памяти
- ▶ C++-подобный компилятор для платформы

Необязательно:

- ▶ Устройства ввода и вывода
- ▶ Сеть
- ▶ Файловая система
- ▶ Дополнительные устройства

Общая модель приложения



Как решать проблемы с железом

- ▶ Разные скорости CPU и объёмы памяти
 - ▶ Включение/отключение возможностей на стадии компиляции
- ▶ Разные размеры экранов и их количество
 - ▶ Задание размера экрана/окна при инициализации/изменении
- ▶ Разные устройства управления
 - ▶ Поддержка в UI layer, возможно частично в Platform layer
- ▶ Разные дополнительные устройства (камера, GPS, etc)
 - ▶ Поддержка в Platform layer, обязательный `#ifdef SOME_DEVICE_SUPPORT`

Как решать проблемы с софтом

- ▶ Разные компиляторы, в том числе и капризные (ADS)
 - ▶ Ответственное использование C++ (namespaces, templates, exceptions, RTTI, global vars, static)
- ▶ Особые методы запуска приложений (BREW, P2K)
 - ▶ Реализация UI как библиотеки
- ▶ Разные API у разных операционных систем
 - ▶ Поддержка в Platform layer

Включение/отключение возможностей на стадии компиляции

```
#define FEATURE_1_SUPPORT
#undef FEATURE_2_SUPPORT
...
// Код feature 1 будет скомпилирован,
// потому что решено поддерживать
// feature 1 на этой платформе.
#ifndef FEATURE_1_SUPPORT
    m_component_loader->LoadFeature1Data();
#endif
...
// Код feature 2 не будет скомпилирован,
// потому что решено не поддерживать
// feature 2 на этой платформе.
// Это уменьшит размер исполняемого кода,
// потребление памяти во время исполнения
// и увеличит его скорость.
#ifndef FEATURE_2_SUPPORT
    m_component_loader->LoadFeature2Data();
#endif
...
```

Как ещё уменьшить размер кода

- ▶ Использовать код повторно, удалять более неиспользуемый код
- ▶ Уменьшить размер временных буферов и/или их время жизни
- ▶ Больше использовать библиотеки платформы
- ▶ Урезать используемые библиотеки
- ▶ Избегать зависимостей от слабых мест компилятора
- ▶ Кросс-компилировать лучшим компилятором
- ▶ Компилировать в более компактный набор инструкций процессора (пример: Thumb для ARM)
- ▶ Сжимать код, в ROM или RAM
- ▶ Использовать плагины/оверлеи
- ▶ Исполнять части кода в другом месте

Ответственное использование C++

- ▶ Стандартные библиотеки не всегда соответствуют стандарту
- ▶ Глобальные переменные (Brew, Symbian)
- ▶ Namespaces (Brew)
- ▶ Exceptions (Symbian)
- ▶ RTTI и `dynamic_cast`
- ▶ 64 бита
- ▶ `signed/unsigned char`
- ▶ Анонимные структуры (ARM)

Ответственное использование C++

- ▶ `for (int i;...)`
- ▶ `#warning (BREW)`
- ▶ Сложные templates
- ▶ Объявление переменной внутри `switch-case`
- ▶ Конвертация цифровых типов вне диапазона меньшего типа (`float -> int`)
- ▶ Отсутствие пустой строки в конце файла
- ▶ `main()` не в C++ файле
- ▶ Регистр имён файлов

Ответственное использование C++

- ▶ long long
- ▶ bool
- ▶ delete(NULL)
- ▶ C++ комментарии в C файлах
- ▶ Не тот символ конца строки
- ▶ Вложенные классы
- ▶ Недублирование virtual
- ▶ Лишние ';' после '{'

Пример платформенного интерфейса

```
// Будет реализовано в платформенном коде.  
class TCPSocket  
{  
    static TCPSocket* New();  
    Status SetListener(TCPSocketListener* listener) = 0;  
  
    Status Connect(const char* host, unsigned short port) = 0;  
    Status Disconnect() = 0;  
    Status Send(const char* data, size_t length, size_t& accepted_length) = 0;  
    Status Recv(char* data, size_t length, size_t& received_length) = 0;  
}  
  
// Будет реализовано в коде ядра.  
class TCPSocketListener  
{  
    Status OnConnected() = 0;  
    Status OnDisconnected() = 0;  
    Status OnDataSent(size_t length) = 0;  
    Status OnDataAvailable(size_t length) = 0;  
    Status OnError(Error code) = 0;  
}
```

Спасибо за внимание

Вопросы?