Тестирование ПО

CMake

Кулаков Кирилл Александрович

Сборка приложений

- Рутинные операции
  - Определение наличия/отсутствия библиотек
  - Определение параметров сборки
  - Типовые операции (сборка объектных файлов, сборка исполняемых файлов)
  - Дополнительные операции (Очистка директории, архивирование, сборка документации)
  - Отслеживание изменений

Ручной способ

- Требуется помнить все имена файлов
- Требуется помнить все ключи компиляции
- Требуется помнить все параметры компилятора
- Нет многопоточности
- Ручная проверка результата

- Удобно для простых / тестовых примеров

Make файл

- Команды для сборки проекта
- Автоматизация отслеживания изменений
- Типовые операции
- Стандарт

- Отсутствие масштабируемости
- Все операции необходимо прописывать вручную

Пример Make файла

```
all: main

main: main.o mylib.o mysublib.o
	g++ main.o mylib.o mysublib.o -o main
main.o: main.cpp
	g++ -c main.cpp
mylib.o: mylib.cpp
	g++ -c mylib.cpp
mysublib.o: mysublib.cpp
	g++ -c mysublib.cpp
clean:
	rm -rf *.o main
```

Генераторы Make файлов

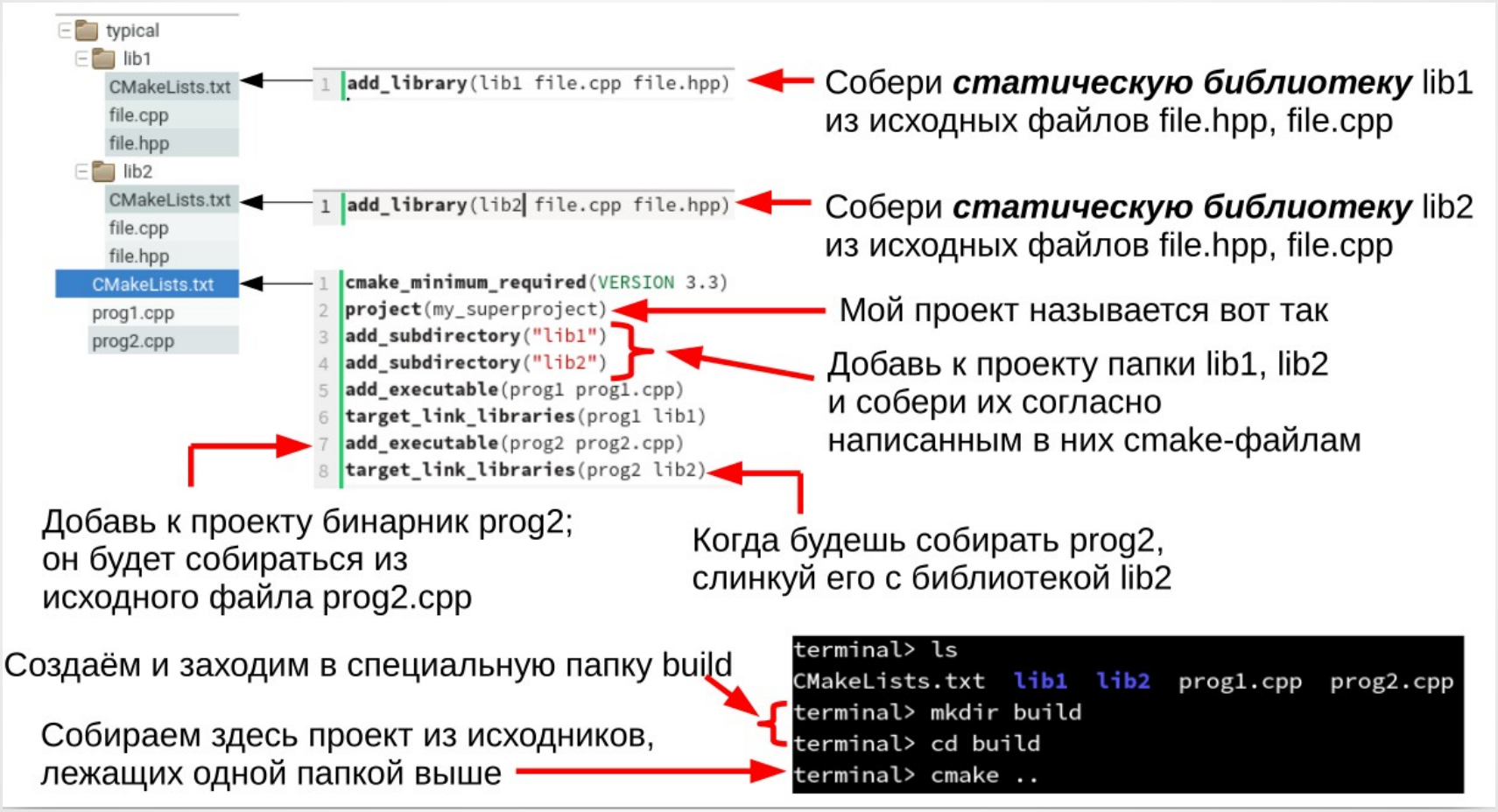
- Использование шаблонов и типовых правил для рутинных задач
- Типовые цели
- Скриптовый язык
- Минимизация кода — пишем только существенные вещи
- Использование в средах разработки
- Теневая сборка (вне дерева исходников)

- Cmake (ninja), QMake, Autotools

CMake

- Файл CMakeLists.txt в каждом каталоге проекта
- Внутри:
  - Набор исходных файлов
  - Их взаимосвязь
  - Особые параметры сборки
  - Флаги компилятора
  - Используемые нестандартные библиотеки
  - Необходимые для сборки требования
  - ...

Пример использования



Теневая сборка + git

- Создаем папку build
- Добавляем build в .gitignore
- В папке build: cmake .. && make

- Исходный код без «мусора»
- Отслеживание изменений исходного кода + изменений CMake файлов
- Проблема с путями до файлов

Особенности CMake

- Использование цели

```
1 add_library(lib file.hpp file.cpp)
2 add_executable(main main.cpp)
3 target_link_libraries(main lib)
4
```

- Последовательное исполнение команд

```
1 cmake_minimum_required(VERSION 3.3)
2 message(">> Started")
3 add_subdirectory("dir")
4 message(">> Subdirectory added")
5 add_executable(main main.cpp)
6 message(">> Finished")
```

```
terminal> cmake .
-- The C compiler identification is GNU 4.8.5
-- The CXX compiler identification is GNU 4.8.5
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - failed
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - failed
>> Started
>> Processing subdirectory
>> Subdirectory added
>> Finished
-- Configuring done
-- Generating done
-- Build files have been written to:
terminal>
```

Команды CMake

- **cmake_minimum_required(VERSION num)** указывается в начале главного cmake-файла; num – номер версии cmake, начиная с которого написанные команды точно заработают
- **add_executable(name src_1 src_2 …)** собрать бинарный файл name из исходников src_i; функция main – в src_1
- **add_dependencies(name dep_1 dep_2 …)** добавить зависимости dep_i к объекту (цели) name
- **add_subdirectory(name)** добавить к проекту папку name и обработать в ней CMakeLists.txt
- **add_library(name src_1 src_2 …)** собрать статическую библиотеку name из исходников src_i
- **target_link_libraries(targ lib_1 lib_2 …)** слинковать объект targ с библиотеками lib_i
- **target_...(targ …)** есть и другие команды, настраивающие отдельные объекты (цели, target'ы)

Команды CMake

- **project(name)**
  - сохраняет имя name в переменную PROJECT_NAME
  - сохраняет путь к исходным файлам в переменные PROJECT_SOURCE_DIR и <name>_SOURCE_DIR
  - сохраняет путь к собираемым файлам в переменные PROJECT_BINARY_DIR и <name>_BINARY_DIR
  - заводит переменные, специфичные для языка исходного кода
  - например, CMAKE_CXX_COMPILER – переменная для компилятора C/C++
  - языки по умолчанию – C и CXX (то есть C++)
- **project(name LANGUAGES lang1 lang2 …)** то же самое, но с явным указанием языков исходного кода lang_i
- **enable_language(lang)** заводит переменные, специфичные для языка lang

Команды CMake

- **set(name val)** установить в переменную name значение val
- **find_package(name)**
  - найти пакет name, поддерживающий сборку системой cmake, и загрузить его объекты и переменные, и результат поиска (да/нет) в <name>_FOUND
  - В дополнительных опциях этой команды можно указать
    - конкретные объекты пакета
    - пути, по которым его требуется искать (помимо стандартных)
    - минимальную требуемую версию пакета

Условные операторы CMake

- if(expr_1)
  - commands

  elseif(expr_2)
  - commands

  elseif(expr_3)
  - commands
  - ...

  else
  - commands

  endif

- foreach(var val_1 val_2 ...)

  command_1

  command_2

  ...

  endforeach(var)

- while(expr)

  commands

  endwhile(expr)

- break(), continue()

Переменные CMake

- **CMAKE_BINARY_DIR** корневая папка, в которой собирается проект
- **CMAKE_SOURCE_DIR** корневая папка, в которой лежат исходные файлы, собираемые cmake’ом
- **CMAKE_INCLUDE_PATH** набор нестандартных папок, в которых cmake будет искать заголовочные файлы (см. include_directories)
- **CMAKE_CURRENT_BINARY_DIR** папка сборки проекта, в которой cmake находится сейчас
- **CMAKE_CURRENT_SOURCE_DIR** папка с исходниками, в которой cmake находится сейчас

Пример CMake

```
cmake_minimum_required(VERSION 3.7)
project(root_or_server_monitor_module)

set(CMAKE_CXX_STANDARD 14)

find_package(PkgConfig REQUIRED)
pkg_check_modules(gtest gtest>=1.10)

add_subdirectory(libormodule2)
add_subdirectory(app)

if (gtest_FOUND)
add_subdirectory(mocks)
add_subdirectory(tests/app-tests)
add_subdirectory(tests/lib2-tests)
add_subdirectory(tests/mqttbrocker-tests)
add_subdirectory(tests/buffer-tests)
add_subdirectory(tests/pqxx-tests)

enable_testing()
add_test(NAME gtest_tests-app COMMAND tests/app-tests/or-server-monitor-module-tests --gtest_output=xml:./or-server-monitor-tests.xml)
add_test(NAME gtest_tests-lib COMMAND tests/lib2-tests/ormodule2-tests --gtest_output=xml:./ormodule2-tests.xml)
add_test(NAME gtest_tests-lib_mqttbrocker COMMAND tests/mqttbrocker-tests/mqttbrocker-tests --gtest_output=xml:./mqttbrocker-tests.xml)
add_test(NAME gtest_tests-lib_buffer COMMAND tests/buffer-tests/buffer-tests --gtest_output=xml:./buffer-tests.xml)
add_test(NAME gtest_tests-pqxx COMMAND tests/pqxx-tests/pqxx-tests --grest_output==xml:./pqxx-tests.xml)

else(gtest_FOUND)
add_subdirectory(mocks-stub)
endif(gtest_FOUND)
```

Пример CMake

```
cmake_minimum_required(VERSION 3.7)

project(ormodule VERSION 2.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_definitions ("-Wall -Wextra -Wpedantic -Weffc++ -Wconversion -Wunreachable-code -Wold-style-cast")

include(FindPkgConfig)

# project files
set(SOURCE_FILES
        broker/IBroker.h
        broker/IBroker.cpp
        broker/IBrokerCallback.h
        broker/DefaultBroker.h
        broker/DefaultBroker.cpp
        broker/MQTTBroker.h
        broker/MQTTBroker.cpp
        database/DatabaseThread.h
        database/DatabaseThread.cpp
        database/TaskStructs.h
        database/IDatabase.h
        database/IDatabase.cpp
        database/IDatabaseCallback.h
        database/DefaultDB.h
        database/DefaultDB.cpp
        database/FileDB.h
        database/FileDB.cpp
        database/Buffer.h
        database/Buffer.cpp
        DynamicDataBlock.h
        DynamicDataBlock.cpp
        ORModule.cpp
        ormodulecallback.cpp
```

Пример CMake

```
cmake_minimum_required(VERSION 3.7)
project(or-server-monitor-module)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_definitions ("-Wall -Wextra -Wpedantic -Weffc++ -Wconversion -Wunreachable-code -Wold-style-cast")

set(SOURCES
        main.cpp
        StorageCallback.cpp
        )

set(HEADERS
        StorageCallback.h
        StatusBlock.h
        ConfigBlock.h
        )

include_directories(${ormodule_SOURCE_DIR})

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

target_link_libraries(${PROJECT_NAME} ormodule)
target_link_libraries(${PROJECT_NAME} uv)

install(TARGETS ${PROJECT_NAME} RUNTIME DESTINATION bin)
```