СИСТЕМИ ЗА ГЕНЕРИРАНЕ НА ПРЕНОСИМ СОФТУЕР

BUILD SYSTEMS FOR GENERATING INDEPENDENT SOFTWARE

Maria Pashinska-Gadzheva
Institute of Mathematics and Informatics - Bulgarian Academy of Sciences,
E-mail: mariqpashinska@math.bas.bg

Abstract
In this paper we present the need for build systems for generating independent software. The main focus of the paper is the CMake build system that is most appropriate for building and managing large scale projects in C/C++. The system allows the development of software independent from the platform. Its main advantages and capabilities are presented in this paper.

Keywords: CMake, build systems

1. ВЪВЕДЕНИЕ

В текущата разработка се представя платформа за създаване на преносим софтуер CMake. Тя е по-различен от стандартните системи за компилиране с това, че не отговаря за самото компилиране на софтуер. CMake създава проект за избрана интегрирана система за разработка или Makefile по първоначално зададени сорс файлове. Така се улеснява разработването на големи софтуерни продукти [1, 2], като акцента пада върху тяхното моделиране и управление. Някои от възможностите, които предоставя CMake, са описани подробно в следващите параграфи:

- CMake се използва за създаване на проекти на езиците C/C++, които от своя страна позволяват програмиране както на ниско, така и на високо ниво. Също така тези езици са създадени за работа и компилиране на различни операционни системи и при различен хардуер.

- Софтуерът е разработен за да бъде свободно достъпен за най-популярните операционни системи. Автоматично се разпознава платформата и наличните необходими компилатори.

- CMake използва собствен език за създаването на проект. Този език разполага със собствени команди и променливи, нужни за описанието на платформата и проекта.

- В текстов файл се описва структурата на проекта и външните библиотеки, необходими за компилирането. Така се изяснява процеса на компилиране и свързване на отделните части на проекта, като същевременно се придобиват умения за правилно моделиране на един софтуерен продукт.

- Дава се възможност за копирането на изпълнимите файлове в нови директории, което позволява да се правят независими промени и да се контролират версиите на разработения софтуер.

- Възможно е избирането на конкретен компилатор, при наличието на няколко такива, като лесно се добавят необходими флагове за компилиране в зависимост от операционната система и избрания компилатор. В резултатния проект, това са допълнителни настройки, за които няма да бъде необходима допълнителна конфигурация на средата.

- Могат да се добавят задължителни компоненти, необходими за паралелно изпълнение, като CMake автоматично открива необходимите файлове.

2. СЪЗДАВАНЕ НА ПРОЕКТ

CMake платформата е разработена първоначално през 1999 [3], като основната идея е да се състави система за създаване на софтуер, който не зависи от операционната система. Тя може да се използва както през терминала на конкретната операционна система, така и чрез графичен интерфейс. Графичният интерфейс позволява наблюдаването на някои променливи и тяхната промяна, както дава по-пълна представа за проекта, който ще бъде създаден. Cmake използва собствен език за описание на структурата на текущия проект. Този език наподобява концепциите на Make файловете и най-вече идеята за използване на цели (targets) [4]. Автоматично открива необходимият компилатор и създава необходимите файлове за работещ проект на избраната интегрирана система. Това позволява да се създадат различни независими проекти спрямо компилатора и интегрираната среда в независими директории. Освен възможността за генериране на независим софтуер, се позволява и автоматизиране на процеса на тестване на един софтуерен продукт, както и неговото пакетиране. В текущата работа ще се спрем върху основните ѝ възможности за създаване на библиотека и изпълними файлове.

За създаването на проект CMake разчита на описанието на зависимостите във файлове със стандартизираното име CMakeLists.txt. Възможно е да се използва единствен текстов файл, намиращ се в директорията на .c или .cpp файловете. Това е приложимо, когато всички необходими файлове са в една директория. Този подход не е общоприет и е неподходящ за големи проекти, включващи в себе си много библиотеки.

При разработването на такъв софтуер, библиотеките и допълнителните .h файловете се намират в различни директории. В този случай се създава текстов файл за всяка директория съдържаща .c или .cpp файлове, които ще бъдат компилирани до обектни файлове. В главната директория се разполага първичен CMakeLists.txt файл, който съдържа задължителните елементи за генерирането на проект. Необходимо е да се добави пътя до останалите текстови файлове. Така се получава дървовидна структура за проекта.

В главния CMakeLists.txt файл се съдържат задължително следните команди:

- cmake_minimum_required(VERSION <number>) - показва минималната задължителна версия на CMake, необходима за генерирането на проекта.

- project(<name>) - задава се името на проекта. Тук може да се добави използваният език, както и версия на самият проект.

- add_subdirectory(<subdir>) - при използване на дървовидна структура на проекта, в тази команда се описват всички поддиректории, съдържащи CMakeLists.txt файлове.

2.1. Конфигуриране и генериране на проект

Създаването на проект чрез CMake има две основни стъпки - конфигуриране и генериране. Преди започване на конфигурирането е необходимо да се окаже директорията, съдържаща главния CMakeLists.txt файл. Възможно е да се даде и директория, в която да бъдат записани и файловете за новосъздадения проект. По подразбиране тази директория съвпада с главната директория. Конфигурирането представлява стартирането на скриптовия файл CMakeLists.txt и откриване на операционната система, необходимия компилатор и нужните файлове. В тази стъпка се задават и стойности на променливи, описващи средата. След конфигурирането, стойностите на някои променливи могат да се променят според нуждите на съответния софтуер, като това се извършва преди генерирането на самия проект. За втората стъпка е необходимо да се избере среда за компилиране на проекта, след което се генерират необходимите файлове. В зависимост от избраната среда е възможно конкретния проект да бъде стартиран от самата система CMake. Двете стъпки могат да се изпълняват както от графичния интерфейс, така и през терминал, като се използва команда **cmake** със съответните опции. Основната разлика е, че при конфигурирането чрез графичния интерфейс, първо се избира средата за компилиране. Главното предимство на графичната среда е лесното наблюдение и промяна на стойностите на променливите.

2.2. Компилиране и свързване

В CMake съществуват две основни функции - за създаване на изпълним файл и за създаване на библиотека. Тези две функции могат да бъдат сравнени с целите (targets) в

Makefile. Под цел се разбира крайния продукт, получен след компилиране. За създаването ѝ се описват всички необходими файлове, както и флаговете на компилатора. В CMake чрез командите **add_executable** и **add_library** се дава име на изходния файл (цел, target) и се посочват всички файлове, от които зависи крайният продукт на съответната команда.

За създаването на библиотека се използва командата **add_library(<name> <list_of_files>)**, чрез която при компилиране на проекта ще се създаде статична или динамична библиотека с оказаното име, като се използват съответните файлове. По подразбиране се създава статична библиотека. Видът на библиотеката може да се окаже след името ѝ. Изходния файл има разширение .a (Linux) или .lib (Windows) за статични библиотеки и .so (Linux) или .dll (Windows) за динамични библиотеки. Създадените библиотеки може да се "инсталират" (запаметят) в специфични за целта директории като се използва командата **install** в CMakeLists.txt. Тази команда позволява копиране на файлове (изпълними, текстови и библиотеки) в оказана директория. За да се изпълни в среда за програмиране следва да се създаде (build) проектът с името INSTALL. При генериране на Makefile за Linux / Unix операционна система от командния ред се изпълнява командата **make install**, който ще запише файловете в посочената дестинация. Също така трябва да се вземе под внимание правата за запис в съответната директория. При създаването на приложение (изпълним файл) се използва командата **add_executable (<name> <list_of_files>)**, за която отново е необходимо създаването на име и списък от файлове. Другата основна команда при създаването на изпълним файл е **target_link_library**, която позволява свързването на библиотеки към изпълним файл.

2.3.Променливи

CMake позволява и дефинирането на променливи чрез командата **set (<VARIABLE_NAME> "<value_1>" "<value_2>" … "<value_n>")**, която задава на стойност на дадена променлива. Тези променливи могат да бъдат дефинирани от потребителя или да бъдат някой от променливите на самата система. Прието е променливите да бъдат изписани с главни букви. За извикване на променлива се използва следното изписване - "${VARIABLE_NAME}". При конфигурирането и генерирането на проекта променливата се заменя с нейната стойност. Следва описание на някои основни променливи, дефинирани в CMake:

- PROJECT_SOURCE_DIR - съдържа директорията на главния CMakeLists файл. Тази и подобни променливи се използват за добавяне на файлове от различни поддиректории, копиране на файлове и др. като позволяват да се използва относителен път до конкретен файл спрямо главната директория на текущия обект. Такива променливи не се показват в графичния интерфейс на системата и не могат да бъдат променяни, тъй като те носят информация за текущия проект.

- CMAKE_CXX_FLAGS - съдържа зададените флагове за C++ компилатора. Тъй като тези флагове зависят от използвания компилатор е подходящо задаването на стойности за тази променлива да се извършва след стартирането на CMake и изпълнението на стъпката конфигуриране. В графичната среда може да се добавят стойности на променливите в зависимост от целевата система, за която се разработва дадения софтуер.

- CMAKE_BUILD_TYPE - съдържа информация за възможните типове за пакетиране на крайния проект. Стандартни типове са Debug и Release. Тези стойности също могат да бъдат променяни от графичната среда.

Съществуват и други видове променливи, сред които са променливите за описване на целевата система, за промяна на поведението на системата. Такива променливи позволяват по-голям контрол върху създаването на проекта. Пълен списък на променливите може да се намери в документацията на системата [6].

2.4. Условен оператор

Различните компилатори използват различни флагове при компилирането. Това е една от причините да не се препоръчва предварителното им задаване. В някои случай обаче е необходимо определени флагове да бъдат използвани за компилирането на програмата или библиотеката. Тук идва ролята на условния оператор. В своя език, подобно на езиците за програмиране, CMake разполага с условен оператор, чрез който може да се сравняват стойностите на различни променливи. Така се позволява създаването на проекти с различни модификации в зависимост от текущата система. В условния оператор могат да се сравняват променливи, съдържащи информация за самата система, както и променливи, които могат да променят своята стойност. В променливата CMAKE_<LANG>_COMPILER, където LANG се замества с конкретния език за програмиране, след сканиране на системата се записва открития компилатор. Тогава тази променлива може да се сравнява със съответните стойности и да се зададат добавят съответните стойности към променливата, съдържаща флаговете.

2.5. Други команди

За създаването на един проект се използват и много други команди. Тук ще бъдат представени няколко команди, които са използвани за генерирането на проект на C++, който създава една статична библиотека и един изпълним файл, използващ тази библиотека.

- target_include_directories(<target_name> OPTION <dir>) - оказва пътя към допълнителни файлове необходими за компилирането на целта. Освен името на целта и директорията на добавените файлове е необходимо и уточняване на начина на използване на допълнителните файлове чрез една от посочените опции - PUBLIC, PRIVATE, INTERFACE, които имат следните значения:

- PRIVATE - допълнителните файлове ще се използват само за текущата цел.
- INTERFACE - допълнителните файлове ще се използват във външни изпълними файлове или библиотеки (цели).
- PUBLIC - файловете могат да се използват както от текущата цел, така и от външни такива.

- source_group(TREE <root_dir> PREFIX <"Group_Name"> FILES <list_of_file_names>) - позволява групиране на файлове в получения проект за интегрираната система. Директорията, посочена от root_dir, е главната директория, в която трябва да се търсят съответните файлове. Group_Name показва името на групата. След FILES трябва да окаже списъка с файлове, които се групират. Тук може да се използва и променлива, дефинирана от **set** функция, която съдържа имената на всички необходими файлове.

2.6. Използване на CMake със системи за паралелно програмиране

CMake може да се използва и с различни системи за паралелно програмиране като MPI, CUDA и OpenMP чрез командата:

find_package(<Package_name> [<version>] [REQUIRED])

Тази команда използва допълнителни файлове за намиране на съответния пакет. Те съдържат информация за пакета и могат да бъдат предоставени като част от него или от самата система CMake. Основна информация, която се намира от **find_package**, е директорията, в която е инсталиран пакета и неговата версия. Ключовата дума REQUIRED задължава системата да има инсталирана съответната платформа, като посочената версия е минималната необходима версия за работа. В противен случай се получава съобщение за грешка. При намирането на необходимите файлове се задават стойности на променливи, съдържащи съответните флагове, библиотеки и директории за правилно компилиране. Повечето системи за паралелно програмиране се използват чрез външни библиотеки. За тях трябва да се добавят допълнителни флагове на компилатора и да се даде директорията, в която е инсталирана библиотеката. Този процес се изразява в допълнително конфигуриране на интегрираната среда. При създаването такъв на проект чрез CMake, тези стъпки се извършват чрез командите **set** (за добавяне на флаговете), **target_include_directories** (за добавяне на директорията на библиотеката) и **target_link_libraries** (за свързване на самата библиотека с изпълнимия файл). Не са необходими допълнителни настройки.

3. РЕЗУЛТАТИ ОТ ПРОВЕДЕНОТО ИЗСЛЕДВАНЕ И ДИСКУСИЯ

Системите за генериране на преносими проекти често не са включени в програмите за обучение в професионално направление 4.6 Информатика и компютърни науки. Това може да доведе до сериозни пропуски в разбирането на методите и технологиите за

разработване на всякакъв вид софтуер. Изучаването на тези системи в каквато и да е форма може не само да обогати знанията по различните изучавани дисциплини, но и да даде практически умения за разработване на софтуерни продукти, които са широко търсени. Интегрирането на тези системи във въвеждащи дисциплини като „Основи на програмирането“ би било довело до подобряване на основата на знанията по програмиране на започващите своето образование. От друга страна създаването и изучаването на курс, обхващащ изцяло тематиката, би развило гореспоменатите знания и умения.

ЗАКЛЮЧЕНИЕ

Системите за генериране на преносим софтуер са мощен инструмент, необходим при създаването на големи проекти. Те позволяват чрез описание на структурата на един проект, той да бъде компилиран върху различни платформи, което от своя страна води до по-лесното му използване. Така се акцентира върху моделирането на самият проект. Процесът на компилиране и свързване на отделните части излиза на преден план. От тази гледна точка могат да бъдат много подходящи при обучението на студенти.

БЛАГОДАРНОСТИ

Работата на автора е частично финансирана от ННП „Млади учени и постдокторанти“ с РМС №577 от 17.08.2018г.

ЛИТЕРАТУРА

[1] J. Elmsheuser et al. (2017). „Large Scale Software Building with CMake in ATLAS“, *Journal of Physics Conference Series*, 898, 072010.
[2] M. Haider, M. Riesch, & C. Jirauschek. (2021). „Realization of best practices in software engineering and scientific writing through ready-to-use project skeletons“. *Opt Quant Electron.* 53, 568.
[3] K. Martin and B. Hoffman. (2007). „An Open Source Approach to Developing Software in a Small Organization“, *IEEE Software*, 24, 46 - 53
[4] P. Smith. (2011). „*Software Build Systems*“, Westford, MA, Pearson Education, Inc.
[5] R. M. Stallman, R. McGrath, P. D. Smith. (2020). *„GNU Make“*, Boston, MA, Free Software Foundation,
[6] https://cmake.org/cmake/help/latest/index.html
[7] https://developer.nvidia.com/cuda-zone