Software Building
(Sestavování aplikací)

Pavel Parízek
parizek@d3s.mff.cuni.cz
Outline

- Gradle
- GNU build system
- CMake
Gradle

- Another popular general-purpose build tool
  - Java, Scala, C, C++, Android
- Encourages best practices (like Maven)
- Script language (DSL) based on Groovy

Web: [https://gradle.org/](https://gradle.org/)

Software Development Tools

Software Building
Gradle – examples

• Scripts
  - https://docs.gradle.org/current/userguide/tutorial_using_tasks.html
  - https://docs.gradle.org/current/userguide/building_java_projects.html

• Running
  - gradle clean build
  - gradle run

• Project template for Java
  - gradle init --type java-application
Motivation for GNU build system

- Portability of programs
  - over different UNIX-like systems
  - existing standards (C, POSIX) define only core aspects

- System-specific configuration
  - e.g., use of KDE instead of Gnome

- Complexity of Make files
  - unreadable, hard to maintain
  - writing all the rules is tedious

- Portability of Make files
  - Make is standardized by POSIX, but not all UNIX-like systems are 100% compliant
Selected portability and compatibility issues

- Programs in C
  - `exit()`: may return `void` or `int (error code)`
  - `free(NULL)`: sometimes does nothing
  - `malloc(0)`: returns `NULL` or valid pointer
  - (and many more)

- Functions in different headers and libraries

- Shell and utilities: Awk, Grep, Sed, ...
  - Multiple implementations (not all compatible)
Solutions for portability and compatibility

- Virtualized environment (Java, C#/.NET)

- GNU build system (Autotools)
  - De-facto standard in Unix/Linux world
  - Explicit support for different flavors
  - Database of known portability issues
  - Resolves issues during configuration
  - Uses only features available everywhere
GNU build system (Autotools)

- Autoconf
  - Configuration detector

- Automake
  - Makefile generator

- Libtool
  - Abstracts creation of libraries

- Gettext
  - Support for localization
End user’s perspective

1. Download the source code
2. “./configure”
   - Automatically tests the target system
     - e.g. for presence of required libraries
   - Detects system configuration (OS, HW)
   - Automatically generates Make files
3. “make”
4. “make install”
End user’s perspective – configuration

- Installation root directory
  - “configure --prefix=/opt”

- Cross-compilation
  - “configure --host”

- Optional features of the software
  - “configure --enable-FEATURE”
  - “configure --disable-FEATURE”

- Optional packages (libraries) to build with
  - “configure --with-PACKAGE”
  - “configure --without-PACKAGE”
What is behind the scenes

shell scripts with many system tests

configure

config.cache

config.log

autoheader

config.h.in

config.h

Makefile.in

Makefile

script which generates make files, taking test results from the cache

automake

Makefile.am

cache for test results between restarts
**Autoconf & the “configure” script**

- Very portable shell script
  - Uses features in the lowest-common-denominator of known shells (no functions, ...)
  - Generated from a template (**configure.ac**)
  - Based on a library of tests of well-known portability and compatibility issues
“configure.ac” script template

AC_INIT(package, version, bug-report-address)

information about the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services

AC_CONFIG_FILES([output file, ...])
AC_OUTPUT
```
AC_INIT([GNU cflow], [1.2], [bug-cflow@gnu.org])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC
AC_PROG_LEX

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h unistd.h locale.h])

AC_OUTPUT
```
“configure” – another example

  - configure.ac
  - src/config.h.in
  - src/client.cpp

- Achieving support for multiple platforms
Generating “configure.ac”

- Autoscan
  - Inspects source files (C/C++) to detect common portability issues
  - Generates skeleton of configure.ac

- Ifnames
  - Reports variables used in preprocessor conditionals
  - Often used to solve platform dependency issues
    - Example: #if HAVE_LOCALE_H
Automake – creating portable Makefiles

configure.ac → automake → Makefile.in

Makefile.am

config.status

Makefile
Supported targets

- install, install-exec, install-data
- uninstall
- clean
- distclean
  - clean to what is distributed
  - removes also files generated by configure
- check
  - run test of compiled binaries
- installcheck
  - run test of installed program
- dist
  - creates source code distribution package (tarball)
“Makefile.am” template

```
SUBDIRS = src

dist_doc_DATA = README

install README into docdir
and put it into distribution
```

```
bin_PROGRAMS = hello

hello_SOURCES = main.c
```

directories to be processed
before this directory
"Makefile.am" template

Makefile.am

\[
\begin{align*}
\text{SUBDIRS} &= \text{src} \\
\text{dist_doc\_DATA} &= \text{README}
\end{align*}
\]

"hello" is a program to be installed into bindir

program "hello" can be built from source "main.c"

since "main.c" is a source file, it will also be put into distribution (by "make dist")

bin\_PROGRAMS = hello
hello\_SOURCES = main.c

src/Makefile.am
Automake – creating libraries

• Static libraries (.a)

```
noinst_LIBRARIES = libgreeting.a
libgreeting_a_SOURCES = greet.c ...

bin_PROGRAMS = hello
hello_SOURCES = main.c
hello_LDADD = libgreeting.a
```

• Shared libraries (.so)

Autotools – the whole picture again

- configure
- config.log
- config.cache
- autoconf
- configure.ac
- ifnames
- autoscan
- autoheader
- config.h.in
- Makefile.in
- automake
- Makefile.am
- Makefile
- config.h
- config.status
- Makefile.in
- Makefile.am
Further reading

- http://www.sourceforge.org/autobook/
- https://www.gnu.org/software/autoconf/
- https://www.gnu.org/software/automake/
CMake

- Cross-platform free and open-source build management application
- Very popular (usage) for programs in C++
- Compiler-independent tool
  - Supports various native build systems (make, Xcode, MS Visual Studio)
- Web: [https://www.cmake.org/](https://www.cmake.org/)
- Two phases of the build process
  - Generate native build scripts from platform-independent configuration (CMakeLists.txt)
  - Run target platform’s native tool for the actual build
Other build tools

- Ivy
  - [https://ant.apache.org/ivy/](https://ant.apache.org/ivy/)

- Scons

- Bazel
  - [http://bazel.io/](http://bazel.io/)
Homework

- Assignment

- Deadline
  - 20.11.2022