ROOT: superbuilds

PAVLO SVIRIN, 2024-05-15

About me

- PhD: National Technical University of Ukraine (2014), Computer Science
- Academic work experience:
  - CERN (2014-2017): project associate at ALICE experiment
  - Brookhaven National Laboratory (2017-2019)
  - CERN (2019-2021): project associate at ATLAS experiment
  - Barcelona Supercomputing Center (2021-2023)
- Speaks Ukrainian, English, Spanish, Chinese, Russian. Some knowledge about Sanksrit, Middle Egyptian, Crimean Tatar.
- Able to write using Cyrillic, Latin, Devanaghari, Georgian, Chinese Simplified alphabets, some Arabic and Hiragana too.

ROOT

- ROOT is a framework for data processing developed at CERN
- Used in high-energy physics and astrophysics
- Provides lots of features for:
  - data processing
  - data saving and data access
  - publish results
  - using interactive sessions using Cling C++ or building customs applications
- Website: https://root.cern/

ROOT: simplification of compilation

- ROOT needs lots of time to compile and user not all of the modules
  - Around 130 internal modules with inter-dependencies
- Practical use case: instead of downloading more then 1GB of full ROOT sources or pre configured ROOT binaries, you can decide to start with minimal set ~150 Mb and expand with any customization you want.

ROOT: simplification of compilation

- The idea is to specify which components have to be compiled during configuration time
- Auto-detection of dependencies among the modules
  - done by parsing of CMakeLists files in search of ROOT_STANDARD_LIBRARY definitions and their dependencies
  - Dependency tracking can be implemented using simple graph database like https://github.com/dpapathanasiou/simple-graph
- Absolutely minimal set of module to be compiled to run ROOT:
  - Core, IO, CLING interpreter, MathCore
  - other modules compiled if specified

ROOT: partial builds

- Goal:
  - to allow to skip compilation of the components which are already built and installed to target directory
  - to easily add new components to distributed modulemap infrastructure
  - in case of admin-only rights to write into ROOT’s installation directory: to install new components together with their modulemap files to different directory and then on ROOT’s start combine all of the necessary modulemaps into one

Distributed modulemap files

- Modulemap in ROOT is a file which defines available components in the installation directory, their headers and shared libraries
- Currently include/module.modulemap a file of several hundreds lines
- We managed to split it into multiple files:
  - each file defines one component
  - main modulemap file just includes all of these files
- Benefits:
  - easy to add new components
  - easy to identify which components are already installed

CMake extenal projects

- A CMake built-in module which allows to decrease the level of coupling among the components in a project
- Can be used as a simple package manager
- https://cmake.org/cmake/help/v3.28/module/ExternalProject.html

```
ExternalProject_Add(secretsauce
  URL               http://intranet.somecompany.com/artifacts/sauce-2.7.tgz
                    https://www.somecompany.com/downloads/sauce-2.7.zip
  URL_HASH          MD5=d41d8cd98f00b204e9800998ecf8427e
  CONFIGURE_COMMAND ""
  BUILD_COMMAND     ${MAKE_EXE} sauce
  DEPENDS tomato onion garlic vinegar
)
```

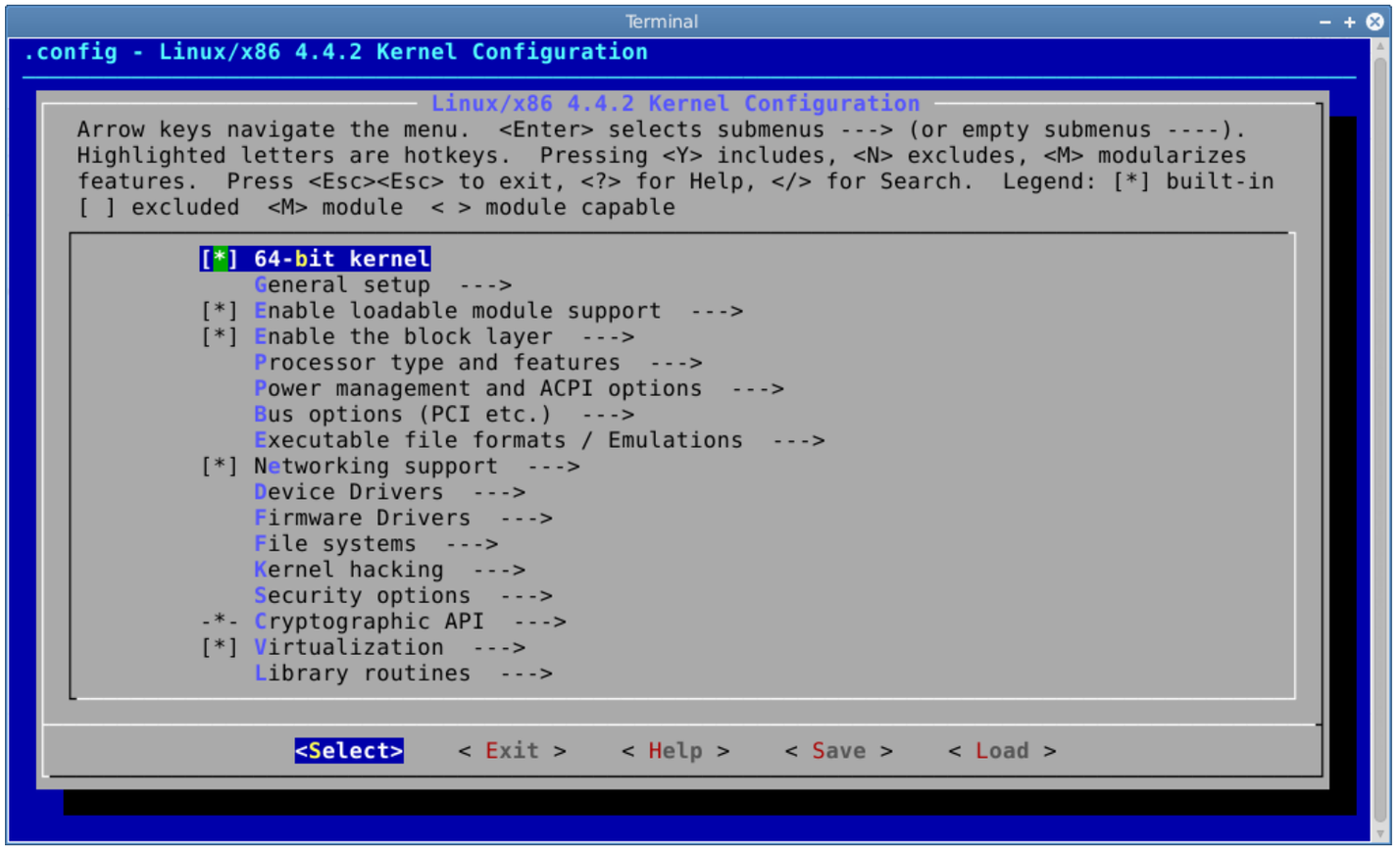
ROOT: menu-based compilation

- Cmake call will look like the following:

```
cmake ../root-6.28.06/ -Dxrootd=0 -Dssl=0 -Dtmva=0 -Dwebgui=0 -Dxproofd=0 -Dgraf=0 -Dexecutables=1
-Dnet=1 -Ddb=1 -Dmath=1 -Dbindings=1 -Dhtml=0 -Dgui=0 -DCMAKE_INSTALL_PREFIX=/mnt/sdb1/opt/
root-modules -Dxml=0 -Dhttp=0 -Dtree=0 -Dproof=0 -Druntime_cxxmodules=1

cmake -DCMAKE_EXTERNAL_PROJECTS=“interpreter;core;io;math” ..
```

- The idea is to develop a similar to Linux’s menuconfig TUI tool which will automatically produce a cmake call from selections
- ncurses
- dialog
- bash-simple-curses

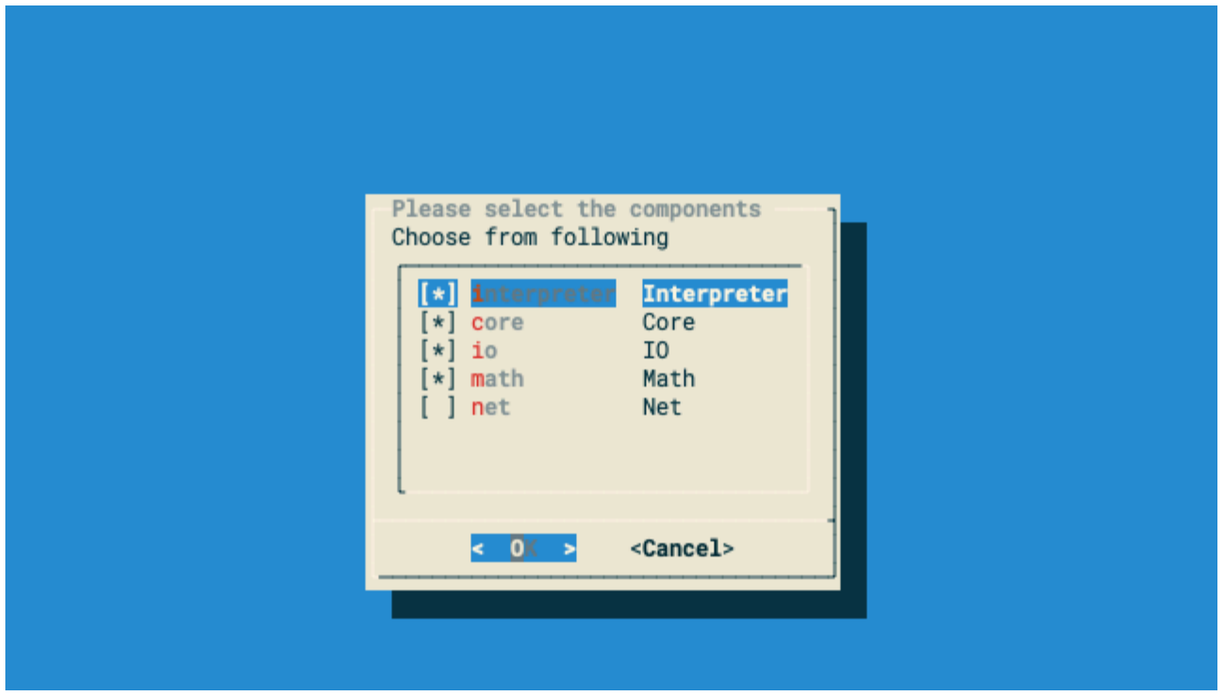


Current status

- External projects defined
- Global configuration step is split into two parts:
  - step on which global variables are defined, then saved to a file
  - step on which external projects load global variables and continue with their own configuration and compilation
- Currently we were able to:
  - configure and build “interpreter” component which has no dependencies on other ROOT components
  - dependencies for other components defined and build processes start in correct order

ROOT: menu-based compilation

```
dialog --output-separator ";" \
      —backtitle "ROOT configutation"  \
      —title "Please select the components " \
      —checklist "Choose from following"  0 0 0 \
            interpreter "Interpreter" on \
            core "Core" on \
            io”IO" on \
            math "Math" on \
            net "Net" off
```




```
>> cat /tmp/c | sed 's/^;//' | xargs -I % echo cmake -DCMAKE_EXTERNAL_PROJECTS=‘"%"' ..

cmake -DCMAKE_EXTERNAL_PROJECTS=“interpreter;core;io;math” ..
```

Closest steps

- Implement cmake modules which discover location of precompiled components in the destination directory, otherwise, looking for them in the root of the source code folder
- update variables in the CMakeLists which reflect the dependencies among external projects