Par-delà la compilation : quelques outils de développement

Journées de la compilation 2019

Pierre Guillou

Dammarie-les-Lys, 31 janvier 2019

(Centre de recherche en informatique, MINES ParisTech, PSL Université Paris)

Plan

Autour de Clang

L’écosystème GCC

```
/usr/bin/c++                                /usr/bin/c89
/usr/bin/c99                                /usr/bin/cc
/usr/bin/cpp                                /usr/bin/g++
/usr/bin/gcc                                /usr/bin/gcc-ar
/usr/bin/gcc-nm                             /usr/bin/gcc-ranlib
/usr/bin/gcov                               /usr/bin/gcov-tool
/usr/bin/x86_64-pc-linux-gnu-c++            /usr/bin/x86_64-pc-linux-gnu-g++
/usr/bin/x86_64-pc-linux-gnu-gcc            /usr/bin/x86_64-pc-linux-gnu-gcc-8.2.
/usr/bin/x86_64-pc-linux-gnu-gcc-ar         /usr/bin/x86_64-pc-linux-gnu-gcc-nm
/usr/bin/x86_64-pc-linux-gnu-gcc-ranlib
```

```
/usr/bin/addr2line   /usr/bin/ar
/usr/bin/as          /usr/bin/c++filt
/usr/bin/dwp         /usr/bin/elfedit
/usr/bin/gprof       /usr/bin/ld
/usr/bin/ld.bfd      /usr/bin/ld.gold
/usr/bin/nm          /usr/bin/objcopy
/usr/bin/objdump     /usr/bin/ranlib
/usr/bin/readelf     /usr/bin/size
/usr/bin/strings     /usr/bin/strip
```

```
pacman -Ql [gcc | binutils] | grep /usr/bin | cut -d' ' -f2
```

L'écosystème Clang

```
pacman -Ql clang | grep /usr/bin | cut -d' ' -f2
```

```
/usr/bin/c-index-test
/usr/bin/clang
/usr/bin/clang++
/usr/bin/clang-7
/usr/bin/clang-apply-replacements
/usr/bin/clang-change-namespace
/usr/bin/clang-check
/usr/bin/clang-cl
/usr/bin/clang-cpp
/usr/bin/clang-format
/usr/bin/clang-func-mapping
/usr/bin/clang-import-test
/usr/bin/clang-include-fixer
/usr/bin/clang-offload-bundler
/usr/bin/clang-query
/usr/bin/clang-refactor
/usr/bin/clang-rename
/usr/bin/clang-reorder-fields
/usr/bin/clang-tidy
/usr/bin/clangd
/usr/bin/diagtool
/usr/bin/find-all-symbols
/usr/bin/git-clang-format
/usr/bin/hmaptool
/usr/bin/modularize
/usr/bin/scan-build
/usr/bin/scan-view
```

Références : Clang Tools [2] et Clang Extra Tools [1]

L'écosystème Clang

```
pacman -Ql clang | grep /usr/bin | cut -d' ' -f2
```

```
/usr/bin/c-index-test
/usr/bin/clang
/usr/bin/clang++
/usr/bin/clang-7
/usr/bin/clang-apply-replacements
/usr/bin/clang-change-namespace
/usr/bin/clang-check
/usr/bin/clang-cl
/usr/bin/clang-cpp
/usr/bin/clang-format
/usr/bin/clang-func-mapping
/usr/bin/clang-import-test
/usr/bin/clang-include-fixer
/usr/bin/clang-offload-bundler
/usr/bin/clang-query
/usr/bin/clang-refactor
/usr/bin/clang-rename
/usr/bin/clang-reorder-fields
/usr/bin/clang-tidy
/usr/bin/clangd
/usr/bin/diagtool
/usr/bin/find-all-symbols
/usr/bin/git-clang-format
/usr/bin/hmaptool
/usr/bin/modularize
/usr/bin/scan-build
/usr/bin/scan-view
```

Références : Clang Tools [2] et Clang Extra Tools [1]

Formatage de code

Compilation source-à-source

indentation insertion d'espaces/tabulations en début de ligne par rapport à la ligne précédente

formatage insertion d'espaces/tabulations/sauts de lignes voire permutation de lignes (includes)

Quelques logiciels de formatage

C/C++/Java (indent), clang-format

Go gofmt

Rust rustfmt

Python autopep8, yapf, black

CMake cmake-format

Clang-format

`clang-format source[.c|.cpp|.h|.hpp]`

- défault : stdout, en place avec `-i`
- `git-clang-format` sur git diff
- fichier de configuration `.clang-format`
- désactivation locale : `/* clang-format off */`

Configuration pour PIPS

```
BasedOnStyle: LLVM
SortIncludes: false
```

Analyse statique de code

`scan-build/scan-view`

- trouver des bugs/mauvaises pratiques sans exécuter le code
  - *unreachable code*
- compilation + rapport navigable (HTML)
- faux positifs

Utilisation

Make `scan-build make [build target]`

Autotools `scan-build ./configure && scan-build make`

CMake `scan-build cmake [sourcedir] && scan-build make`

Meson `ninja scan-build`

Démo scan-build/scan-view i

```c
#include <stdio.h>
#include <stdlib.h>

void malloc_sizeof(void) {
  int *a = (int *)malloc(sizeof(unsigned int));
  int *b = (int *)malloc(sizeof(int *));
  free(a);
  free(b);
}

void use_after_free(void) {
  int *a = (int *)malloc(sizeof(int));
  free(a);
  printf("%d\n", *a);
}

void uninitialized_leak(void) {
  int *a = (int *)malloc(sizeof(int));
  if (*a)
    printf("hello\n");
}
```

Démo `scan-build/scan-view` ii

```c
void bad_free(int *a) { free(a); }

int dead_increment(void) {
  int a = 1;
  return a++;
}

void malloc_zero(size_t n) {
  int *a = (int *)malloc(n * sizeof(int));
  printf("%d\n", a[n - 1]);
}

int main(void) {
  int a[2];
  bad_free(a);
  size_t b = 0;
  malloc_zero(b);
  return 0;
}
```

Clang-tidy

- trouve des défauts dans le code
- défauts parfois aussi relevés par `scan-build`
- peut en résoudre certains (*fixits*)
- `clang-tidy -p builddir [-checks=list] source[.c|.cpp]`
- nécessite une compilation database

Outils similaires

Python	flake8, pylint
Rust	clippy
JavaScript	eslint
CMake	cmake-lint
LaTeX	chktex

Compilation database ?

- `compile_commands.json`
- commande de compilation pour chaque fichier source
- à la racine du dossier de build
- générée via

Make	`bear make`
CMake	`SET(CMAKE_EXPORT_COMPILE_COMMANDS TRUE)`
Meson	généré automatiquement

```
[
  {
    "directory": "/home/pierre/papers/jcompil19/assets/code/sb/build",
    "command": "ccache c++ -Isb@exe -I. -I.. -fdiagnostics-color=always
    ↪  -pipe -D_FILE_OFFSET_BITS=64 -Wall -Winvalid-pch
    ↪  -Wnon-virtual-dtor -g  -MD -MQ 'sb@exe/sb.cpp.o' -MF
    ↪  'sb@exe/sb.cpp.o.d' -o 'sb@exe/sb.cpp.o' -c ../sb.cpp",
    "file": "../sb.cpp"
  }
]
```

Clang-tidy : catégories des lints

Name prefix	Description
boost-	Checks related to Boost library.
bugprone-	Checks that target bugprone code constructs.
cert-	Checks related to CERT Secure Coding Guidelines.
cppcoreguidelines-	Checks related to C++ Core Guidelines.
clang-analyzer-	Clang Static Analyzer checks.
google-	Checks related to Google coding conventions.
hicpp-	Checks related to High Integrity C++ Coding Standard.
llvm-	Checks related to the LLVM coding conventions.
modernize-	Checks that advocate usage of “C++11” language constructs.
mpi-	Checks related to MPI (Message Passing Interface).
performance-	Checks that target performance-related issues.
portability-	Checks that target portability-related issues
readability-	Checks that target readability-related issues
...	...

Les *Sanitizers* : analyse dynamique

- instrumentation de code à la compilation
- run-time léger
- GCC et Clang
- 5×–20× plus rapide que Valgrind
- sortie en couleur

Nom	Flag	Objet
AddressSanitizer	-fsanitize=address	Accès mémoire invalides
└ LeakSanitizer	-fsanitize=address	Fuites mémoire
UndefinedBehaviorSanitizer	-fsanitize=undefined	Comportements indéfinis
ThreadSanitizer	-fsanitize=thread	Accès concurrents
MemorySanitizer	-fsanitize=memory	Mémoire non initialisée (Clang)

Le protocole *Language Server*

Éditeurs de code : Emacs

Fonctions de base : un plugin par langage

- coloration syntaxique
- vérification de la syntaxe (parfois)
- auto-complémentation *contextuelle*
- indentation

Fonctions avancées : plugins spécifiques

auto-complémentation *sémantique*	company vs ac
trouver la déclaration	ctags/etags
documentation, types	eldoc
diagnostics	flymake vs flycheck
formatage	
surlignage des variables	

Défauts

- duplication des fonctionnalités entre éditeurs
- mise en place par les utilisateurs ?
- remontée des infos de configuration ?
- intégration des fonctions pour différents langages ?

Défauts

- duplication des fonctionnalités entre éditeurs
- mise en place par les utilisateurs ?
- remontée des infos de configuration ?
- intégration des fonctions pour différents langages ?

$\longrightarrow$ LSP

Le protocole *Language Server*

but	étendre les éditeurs de texte $\longrightarrow$ IDE
protocole	JSON-RPC défini par Microsoft en 2016 [3]
clients	éditeurs de texte
serveurs	spécifiques au langages de programmation
configuration	données issues d'une compilation database

VSCode

Exemple de serveurs

C/C++	clangd, cquery, ccls
Python	pyls
Rust	rls
Bash	bash-language-server

mais aussi Java, PHP, Go, JS, TypeScript, Ruby, XML, YAML... [4]

Avantages

Sans LSP : 1 plugin/langage/éditeur(/fonction)

Éditeurs	Langages			
	C/C++	Python	Rust	...
Vim	✓	✓	✓	...
Emacs	✓	✓	✓	...
VSCode	✓	✓	✓	...
...	?	?	?	?

Avec LSP : 1 plugin/éditeur, 1 serveur/langage

Éditeurs	Client LSP
Vim	✓
Emacs	✓
VSCode	✓
...	?

Langages	Serveur LSP
C/C++	✓
Python	✓
Rust	✓
...	?

Des systèmes de compilation

Systèmes de compilation

Configuration et générateurs

- réagir à la présence/absence de dépendences sur l'hôte
- activer/désactiver des fonctionalités
- télécharger/compiler certaines dépendences — sous-modules
- générer des règles de compilation — Make, Ninja

Les plus courants

Make	simple à petite dose
Autotools	complexe à mettre en œuvre
CMake	standard de fait (pour le C++)
Meson	« nouvelle » (2013) alternative

CMake vs Meson : syntaxe

```
cmake_minimum_required(VERSION 3.13)
project(Tutorial)

add_library(MyLib mylib/mylib.c)

add_executable(Tutorial tutorial.c)
target_link_libraries(Tutorial MyLib)
target_include_directories(Tutorial PRIVATE mylib)
```

```
project('Tutorial', 'c')

mylib = library(
  'mylib',
  'mylib' / 'mylib.c',
)

tutorial = executable(
  'Tutorial',
  'tutorial.c',
  include_directories : include_directories('mylib'),
  link_with : mylib,
)
```

« J'aime pas CMake » vs Meson

- syntaxe
- langage Turing-complet
- warnings compilateur en N&B
- `cmake [sourcedir|builddir]`
  - fixed in CMake 3.13 (12/2018)
- gestion des dépendances/sous-projets
- documentation
- pas de support out-of-the-box de :
  - ccache
  - scan-build
  - sanitizers
  - compilation database
  - …

« J’aime bien Meson, mais… »

- lister les fichiers source — *pas de globbing*
- générateurs à sortie unique — *Swig*
- pas (encore) de GUI — *ni de ncurses*
- gestion des dépendances CMake

Références i


Clang Extra Tools. URL : https://clang.llvm.org/extra/index.html.


Clang Tools Overview. URL : https://clang.llvm.org/docs/ClangTools.html.


Language Server Protocol. URL : https://microsoft.github.io/language-server-protocol/.


Language Server Protocol implementations. URL : https://langserver.org/.