An Analysis of Library Usage in the C++ Code Base of Fedora Linux 37

Jiachao Deng and Michael D. Adams

2024-10-29

Outline

Challenges of Large-Scale C++ Code Analysis


```cpp
#include <iostream>

int main(int argc,
  char const *argv[]) {

  std::cout << "Hello, World!\n";
  return std::cout.flush() ? 0 : 1;
}
```

Listing 1: A hello world program in C++

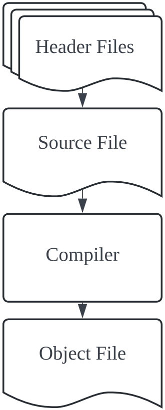

Figure 1: Translating a C++ source file to a binary file.

Outline

2 Background

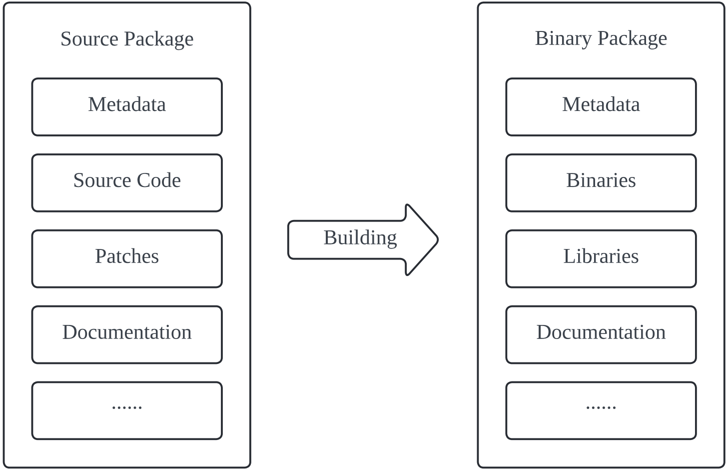

Figure 2: A binary package built from a source package.

```cpp
#include <iostream>

int main(int argc,
  char const *argv[]) {

  std::cout << "Hello,
      World!\n";
  return std::cout.flush() ? 0
      : 1;
}
```

Listing 2: A hello world program.

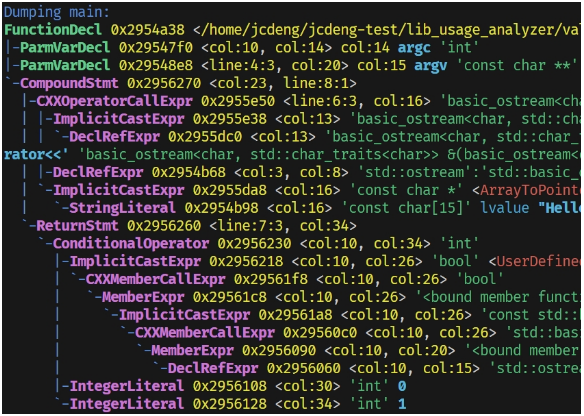

Figure 3: Abstract language tree of the hello world program.

Compiler Flags


```cpp
#include <iostream>

#ifdef ENABLE_FOO_FEATURE
void foo() { std::cout << "Foo feature enabled\n"; }
#endif

int main() {
    #ifdef ENABLE_FOO_FEATURE
    foo();
    #else
    std::cout << "Foo feature not available.\n";
    #endif
    return 0;
}
```

Listing 3: Code blocks toggled by a compiler flag.

Build Systems and Build Wrapper

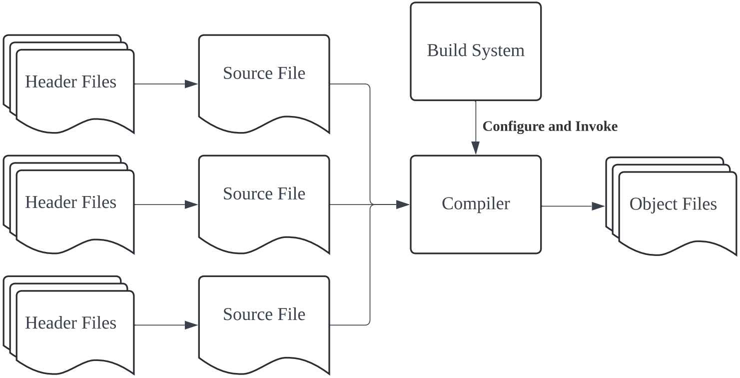

Figure 4: A build system orchestrates the compilation of source files.

The `bear` Build Wrapper

2 Background

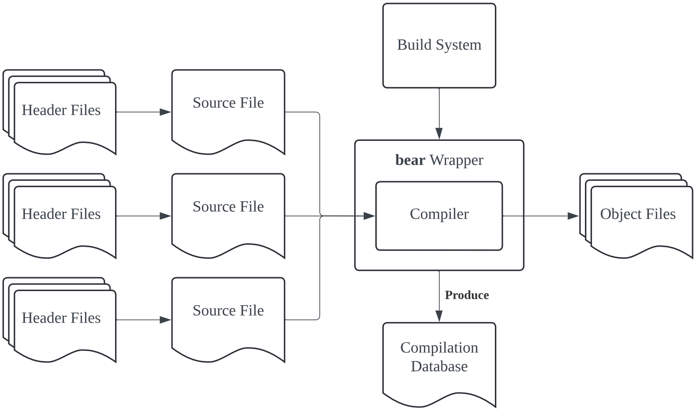

Figure 5: The `bear` build wrapper captures compiler flags and outputs a compilation database.

Software Dependency

2 Background

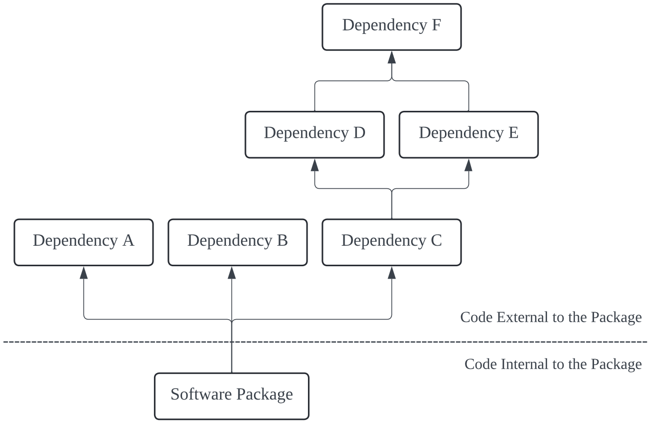

Figure 6: Software dependencies must be installed before building a package.

Software Repositories and the `dnf` Package Manager

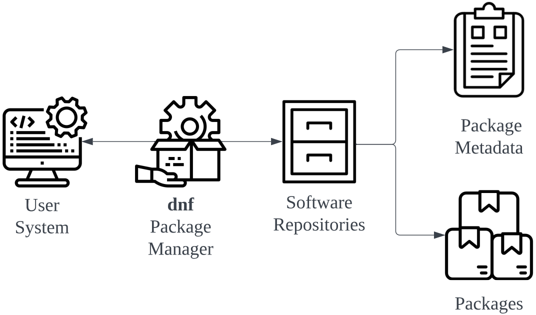

Figure 7: Obtain software packages from software repositories using the `dnf` package manager.

Outline

Functionalities of the Package Processing Framework

Software package selection:

- Source packages that depend on the `gcc-c++` package.
- Source packages that depend on the `clang` package and have source files with C++ file extensions.

Information that must be prepared for each source package:

- Any header files internal/external to the package that are included by the one or more C++ source files in the package.
- All C++ source files in the package that would be compiled during the build process, including generated source files.
- Compiler flags used to compile each C++ source file in the package.

Package Processing Steps

The framework has four major steps to process a source package:

- Prebuild. Before building a package, install all dependencies of the package and extract package contents.
- Build. Build the package and capture compiler flags using a build wrapper.
- Postbuild. Clean up unnecessary files for source code analysis.
- Analysis. Apply the analysis tool to package source code.

Outline

Definition of the Library Usage

4 Proposed Analysis Tool

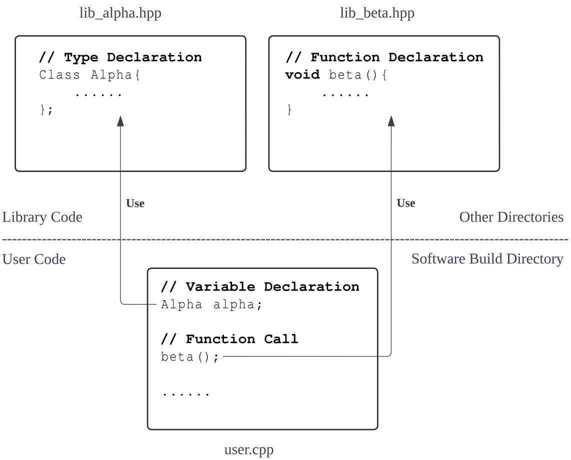

Figure 8: A package source file uses entities declared in library locations.

Examples of Library Usage


```
#include <iostream>

int main(int argc,
char const *argv[]) {

  std::cout << "Hello, World!\n";
  return std::cout.flush() ? 0 : 1;
}
```

Listing 4: Library usages in the hello world program.

Outline

5 Results of Applying Framework

Results of Package Processing

5 Results of Applying Framework

Table 1: Package processing outcomes

Outcome	Number of Packages	% of Packages
prebuild failed	17	0.57
build completed but failed	196	6.58
build failed due to hanging	31	1.04
postbuild failed	11	0.37
$[0\%, 50\%)$ analysis success	84	2.82
$[50\%, 100\%)$ analysis success	372	12.48
100% analysis success	2269	76.14
total	2980	100.00

Source Package Sizes

Table 2: Distribution of processed source code size for packages with at least one successfully analyzed C++ source file

Lines of Code	Number of Packages	% of Packages
$[1, 10)$	7	0.29
$[10, 100)$	6	0.25
$[100, 1000)$	201	8.45
$[1000, 10000)$	645	27.11
$[10000, 100000)$	1018	42.79
$[100000, 1000000)$	448	18.83
$[1000000, 10000000)$	49	2.06
$[10000000, 100000000)$	5	0.21
total	2379	100.00

Largest Source Packages Processed

Table 3: 10 packages with the most lines of C++ code

Name of Source Package	Number of Lines
`libint2-0:2.6.0`	28724722
`swig-0:4.0.2`	14987941
`godot-0:3.4.5`	14451968
`qt5-qtwebengine-0:5.15.10`	12959278
`cross-gcc-0:12.1.1`	10843595
`nextpnr-0:1-12.20220912gitf1349e1`	10763584
`root-0:6.26.06`	7421955
`libint-0:1.2.1`	7094630
`swift-lang-0:5.7`	7004290
`llvm9.0-0:9.0.1`	6662584

Outline

6 Results of Library Usage Analysis

Standard Library Algorithms

Table 4: Most frequently used STD algorithms

Function Name	Appear in % of Packages
min	40.31
max	37.24
sort	34.59
find	29.80
copy	22.36
find_if	20.47
transform	20.39
reverse	13.37
fill	12.90
lower_bound	12.40

Table 5: Least frequently used STD algorithms

Function Name	Appear in % of Packages
remove_copy	0.25
partition_copy	0.21
prev_permutation	0.17
replace_copy_if	0.13
rotate_copy	0.08
replace_copy	0.08
search_n	0.04
is_sorted_until	0.04
is_partitioned	0.04
sample	0.04

Bounds-Checking When Indexing

Table 6: Fraction of indexing operations without built-in bounds-checking when using sequential container/view types

Type Name	% of Indexing Operations
array	95.54
basic_string	91.96
vector	91.69
deque	90.91
basic_string_view	87.52

Table 7: Fraction of packages using only indexing operations without built-in bounds-checking when using sequential container/view types

Type Name	% of Packages
array	86.80
basic_string_view	85.42
vector	77.20
basic_string	74.57
deque	63.01

University of Victoria

An Analysis of Library Usage in the C++ Code Base of Fedora Linux 37

Thank you for listening!