Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Реинжиниринг системы сборки Mybuild для Embox

Михайлов Илья Игоревич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования, Косарев Д. С.

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. Mybuild 5

2.2. Инструменты 9

3. Реализация **10**

3.1. Проектирование АСТ 10

3.2. Лексический и синтаксический анализ 11

3.3. Результат работы программы 11

4. Тестирование **13**

Заключение **14**

Список литературы **15**

Введение

Системы сборки — это неотъемлемая часть разработки программного обеспечения, так как позволяет автоматизировать некоторые однотипные задачи, освобождая ресурсы на что-то другое. Системы сборки также применяются для создания конкретных образов конфигурируемых ОС, которые включают необходимые для какой-либо задачи модули. Представителем такого класса ОС является ОСРВ (операционная система реального времени) Embox, которая использует систему сборки Mybuild.

Можно сформулировать задачу системы сборки так: собрать актуальную версию проекта на основе зависимостей между его частями и данных об их изменении, выполнив минимум задач по перекомпилированию [3]. Главный пример системы сборки — это GNU Make, написанный на языке C и повсеместно использующийся на Unix-подобных ОС. Результатом работы некоторых систем сборки являются make-файлы, что позволяет повысить уровень абстракции при описании зависимостей. Примером таких систем являются CMake, Meson и Mybuild.

Реализация Mybuild написана на языке GNU Make, из-за чего возникают проблемы с масштабированием данной системы. Также код плохо задокументирован, в следствие чего работать с данным проектом становится затруднительно.

Предлагается провести реинжиниринг системы сборки Mybuild, а именно написать реализацию на функциональном языке OCaml.

1. Постановка задачи

Целью работы является реинжиниринг системы сборки Mybuild. Для её выполнения были поставлены следующие задачи:

- осенний семестр:
    1. реализовать парсер декларативного языка Myfile;
    2. протестировать парсер языка Myfile на существующей кодовой базе;
- весенний семестр:
    1. реализовать парсер декларативного языка Configfile;
    2. реализовать генерацию make-файлов и других необходимых компонентов для сборки Embox;
    3. провести тестирование.

2. Обзор

2.1. Mybuild

2.1.1. Обзор

Embox является модульной и настраиваемой ОСРВ. Для описания таких модулей и конфигураций используется система сборки Mybuild, которая генерирует make-файлы и остальные необходимые для создания образа файлы. Система полностью написана на языке GNU Make и состоит из следующих частей:

- классические make-файлы, к которым в основном относятся скрипты раскрутки;
- библиотеки функций, т.е. make-файлы, написанные на синтаксическом расширении языка GNU Make, где находится большая часть логики системы сборки;
- вспомогательные скрипты, которые могут порождать дополнительные файлы в образе сборки.

2.1.2. Процесс сборки

Процесс создания образа ОС с помощью Mybuild можно разбить на следующие стадии:

1. подготовка скриптов;
2. создание графа описания модулей;
3. создание модели системы;
4. генерация необходимых ресурсов;
5. запуск скриптов на исполнение.

2.1.3. Подготовка скриптов

Так как скрипты написаны на синтаксическом расширении языка GNU Make, необходимо преобразовать их с помощью препроцессора. Затем, чтобы избежать повторного запуска препроцессора на всех скриптах после небольших изменений, используется скрипт кэширования (mk/script/mk-cache.mk)

Листинг 1: Пример определения класса

```
define class_Foo
    $(field message,$1)
    $(method sayHello,
        $(info $(get_field message)))
endef
define some_func
    $(for foo <- $(new Foo,Privet!),
        $(invoke foo->sayHello)
        # Return the instance.
        $(foo))
endef
```

2.1.4. Языки описания

Mybuild позволяет описывать проект с помощью двух декларативных объектно-ориентированных языков программирования:

- Myfile — для описания модулей;
- Configfile — для описания конфигураций.

Данные языки имеют схожий синтаксис, который вдохновлен Java.

Для синтаксического разбора декларативных языков используется генератор LALR-парсеров [2] GOLD Parsing system [1], который так же, как и Mybuild, написан на GNU Make. Грамматика для языка в данной утилите описывается с помощью своего диалекта EBNF (Extended

[1] http://www.goldparser.org/ (дата обращения: 08.01.2024)

Backus-Naur Form) [2]. Следует отметить, что последняя версия генератора
(GOLD 5.2.0) была выпущена 18-го августа 2012 года, что означает, что данный инструмент является устаревшим и больше не поддерживается.

2.1.5. Создание графа описания модулей

Можно получить граф описания модулей в результате следующей последовательности действий:

1. поиск всех .my-файлов и Myfile'ов в директории с исходным кодом;
2. синтаксический разбор с помощью GOLD, и на основе разбора построение модели файла (mk/mybuild/myfile-model.mk);
3. объединение всех моделей в граф;
4. проведение необходимых проверок (на ацикличность и т.д.);
5. кэширование полученного набора моделей.

2.1.6. Создание модели системы

Далее происходит синтаксический разбор конфигурационных файлов, где в каждом из них описаны необходимые для конкретной сборки модули, и построение их моделей. Затем происходит построение Build-модели (mk/mybuild/build-model.mk), куда входят модули, их файлы исходного кода, опции модулей и компиляции. Для каждого модуля в Build-модели необходимо:

- включить сам модуль;
- включить все его зависимости и провести проверку (например, включить реализацию по-умолчанию для абстрактного модуля, если никакой другой реализации не было включено);

[2] http://www.goldparser.org/doc/grammars/index.htm (дата обращения: 08.01.2024)

- сопоставить значениям аргументов, указанных в конфигурационных файлах, значения option'ов модуля, и если значение аргумента не указано, установить значение по-умолчанию.

После данного процесса необходимо провести итоговую проверку на корректность Build-модели.

2.1.7. Генерация необходимых ресурсов

За генерацию необходимых ресурсов отвечает mk/script/build/build-gen.mk, который порождает:

- исходные файлы на языке C, содержащие run-time представление Build-модели;
- заголовочные файлы с необходимыми опциями; также происходит копирование исходных заголовочных файлов модуля;
- командные файлы для файлов с исходным кодом, содержащие параметры командной строки компилятора;
- make-файлы с правилами для сборки целевого образа и промежуточных объектных файлов.

2.1.8. Пример описания простейшего модуля

Чтобы продемонстрировать работу Mybuild, в официальной документации [3] есть следующий пример добавления модуля в образ ОС:

Листинг 2: Пример описания модуля

```
package embox.cmd
@AutoCmd
@Cmd(name = "hello", help="First Embox application")
module hello {
    source "hello.c"
}
```

[3] https://github.com/embox/embox-docs/releases/download/latest/embox_quick_start_ru.pdf (дата обращения: 08.01.2024)

Листинг 3: Пример описания конфигурации

```
package genconfig
configuration conf {
    // ...
    include embox.cmd.tutorial.hello
}
```

Создав директорию ./src/cmds/hello, добавив в нее hello.c и файл Mybuild с описанием модуля (Листинг 2), и добавив данный модуль в необходимую конфигурацию (Листинг 3), можно собрать образ системы (например, для x86-qemu) и запустить тестовый модуль с помощью:

Листинг 4: Запуск тестового модуля

```
    make confload-x86/qemu
    make
    ./scripts/qemu/auto_qemu
```

2.2. Инструменты

2.2.1. Menhir и ocamllex

В качестве инструментов для построения парсера были выбраны генератор лексических анализаторов ocamllex и генератор синтаксических анализаторов Menhir. Menhir является генератором LR(1) парсеров [4], что означает, что такой парсер способен распознавать LALR(1)-языки. Следовательно такой генератор парсеров подходит для задачи.

Также был рассмотрен вариант использования библиотеки парсер-комбинаторов, например Angstrom [4]. Хоть парсеры-комбинаторы и считаются более выразительным инструментом для синтаксического анализа [1], для данной данной задачи разбора декларативного LALR(1)-языка по уже существующей грамматике будет удобнее использовать генератор парсеров.

[4] https://github.com/inhabitedtype/angstrom (дата обращения: 08.01.2024)

3. Реализация

Грамматики декларативных языков Myfile и Configfile описаны в формате EBNF. Далее по .grm файлам[5] GOLD генерирует необходимый парсер.

Листинг 5: Пример описания грамматики

```
<ModuleMembers>
        ::= <AnnotatedModuleMember> <ModuleMembers>
          |
<AnnotatedModuleMember>
        ::= <Annotations> <ModuleMember>
<ModuleMember>
        ::= 'depends'  <ReferenceList>
          | 'provides' <ReferenceList>
          | 'requires' <ReferenceList>
          | 'source'   <FilenameList>
          | 'object'   <FilenameList>
          | 'option'   <Option>
```

3.1. Проектирование АСТ

Для создания своей реализации парсера необходимо в первую очередь спроектировать структуру данных для АСТ (абстрактного дерева синтаксиса). Для этого использовались алгебраические типы данных OCaml, а также библиотека deriving, для облегчения печати дерева. Во время проектирования необходимо было избавляться от некоторых избыточных правил, а также нужно было переименовать названия нетерминалов для лучшей читабельности.

Листинг 6: Пример структуры данных из ast.mli

```
type module_member =
  | Depends of composite_names
```

[5] https://github.com/embox/embox/blob/master/mk/mybuild/myfile.grm (дата обращения: 08.01.2024)

```
  | Provides of composite_names
  | Requires of composite_names
  | Source of file_names
  | Object of file_names
  | Opt of definition
[@@deriving show { with_path = false }]

type module_members = (annotations  * module_member) list
```

3.2. Лексический и синтаксический анализ

Menhir и ocamllex были использованы для синтаксического и лексического анализа соответственно. Используемые наборы символов были взяты из myfile.grm файла и перенесены в описание лексера. Для токенизации строк и игнорирования комментариев были использованы рекурсивные лексеры. Правила из myfile.grm файла переносились в описание парсера.

Листинг 7: Пример описания правил в parser.mly

```
annotated_module_member:
  | annotations = annotation*; module_member = module_member {
    annotations, module_member };
module_member:
  | DEPENDS_KW; references = references { Depends references }
  | PROVIDES_KW; references = references { Provides references }
  | REQUIRES_KW; references = references { Requires references }
  | SOURCE_KW; filenames = filenames { Source filenames }
  | OBJECT_KW; filenames = filenames { Object filenames }
  | OPTION_KW; definition = definition { Opt definition };
```

3.3. Результат работы программы

Результат работы парсера на одном из файлов Mybuild.

```
$ ./_build/default/src/parser/parse.exe ../embox/src/mem/misc/
   Mybuild
((Some ["embox"; "mem"]), [],
 [([],
   (Module
      ([], "slab", None,
       [([], (Opt (NumberType, "heap_size", (Some (
          NumberLiteral 524288.)))));
        ([], (Source ["slab.c"; "slab_impl.h"]));
        ([], (Depends [["embox"; "mem"; "page_api"]]));
        ([], (Depends [["embox"; "mem"; "phymem"]]));
        ([], (Depends [["embox"; "mem"; "heap_place"]]))])));
   ([(["DefaultImpl"], (Some (Value (Refer ["pool_ndebug"]))))
     ],
    (Module ([Abstract], "pool", None, [])));
   ([],
    (Module
       ([], "pool_ndebug", (Some ["pool"]),
        [([], (Source ["pool.c"]));
          ([], (Depends [["embox"; "util"; "SList"]]))])));
   ([],
    (Module
       ([], "pool_debug", (Some ["pool"]),
        [([], (Source ["pool_debug.c"])); ([], (Source ["
           pool_debug.h"]));
          ([], (Depends [["embox"; "util"; "SList"]]));
          ([], (Depends [["embox"; "util"; "Bitmap"]]))])))
   ])
```

4. Тестирование

Парсер

Парсер был протестирован на файлах Mybuild из репозитория Embox используя cram-тесты [6].

Непрерывная интеграция

С помощью Github Actions была реализована непрерывная интеграция, где происходит сборка проекта при помощи dune, а также запуск тестов.

[6] https://github.com/mikhaylovilya/caravan/blob/master/test/parser_test/parse_myfiles.t (дата обращения: 08.01.2024)

Заключение

В результате проведенной работы были выполнены следующие задачи:

1. написана реализация парсера декларативного языка Myfile;
2. проведено тестирование парсера на существующей кодовой базе в репозитории Embox.

Исходный код работы доступен по ссылке [7]. Имя аккаунта: mikhaylovilya. Название репозитория: caravan.

[7] https://github.com/mikhaylovilya/caravan (дата обращения: 08.01.2024)

Список литературы

[1] Efficient parsing with parser combinators / Jan Kurš, Jan Vraný, Mohammad Ghafari et al. // Science of Computer Programming. — 2018. — Vol. 161. — P. 57–88. — Advances in Dynamic Languages. URL: https://www.sciencedirect.com/science/article/pii/S0167642317302654 (дата обращения: 08.01.2024).

[2] Grune Dick. Parsing Techniques: A Practical Guide. — 2nd edition. — Springer Publishing Company, Incorporated, 2010. — ISBN: 1441919015.

[3] Mokhov Andrey, Mitchell Neil, Peyton Jones Simon. Build systems a la carte // Proc International Conference on Functional Programming (ICFP'18). — ACM, 2018. — September. — URL: https://www.microsoft.com/en-us/research/publication/build-systems-la-carte/ (дата обращения: 08.01.2024).

[4] Pottier François, Régis-Gianas Yann. Menhir Reference Manual (version 20231231). — 2021. — URL: https://gallium.inria.fr/~fpottier/menhir/manual.html (дата обращения: 08.01.2024).