ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ

«СКАДИ»

(ООО «СКАДИ»)

УТВЕРЖДАЮ

КОМПИЛЯТОР ЯЗЫКА МУЛЬТИОБЕРОН СО СМЕННЫМИ БЭКЕНДАМИ

Описание программы

34185873.425510.003.ОП.80.М

(На 30 листах)

2023

РАЗРАБОТАЛ

СОДЕРЖАНИЕ

1 ОБЩИЕ СВЕДЕНИЯ 4
1.1 Обозначение и наименование программ 4
1.2 Программное обеспечение, необходимое для функционирования программы 4
1.3 Языки программирования, на которых написана программа 4
2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ 5
2.1 Классы решаемых задач 5
2.2 Назначение программы и основные функции 5
2.3 Сведения о функциональных ограничениях на применение 6
3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ 8
3.1 Основные алгоритмы программ 8
3.2 Используемые методы трансформации 9
3.3 Структура и составные части ПО 11
3.4 Связи с другими программами 12
4 ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА 13
5 ВЫЗОВ И ЗАГРУЗКА 14
5.1 Динамически загружаемые модули 14
5.2 Точки входа в программу 14
6 ВХОДНЫЕ ДАННЫЕ 16
6.1 Дженерики и препроцессор 16
6.2 Операции ограничения 16
7 ВЫХОДНЫЕ ДАННЫЕ 18
7.1 Примеры текстовой генерации выходных файлов 18
7.2 Разработка платформо-зависимых модулей 18
ПРИЛОЖЕНИЯ 20
Приложение 1 – Состав каталогов МультиОберон 20
Приложение 2 – Пример Hello World 22
Приложение 3 – Сгенеренный C-код для Hello World 23
Приложение 4 – Текстовый LLVM-код для Hello World 25
ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ 29
ЛИСТ ИЗМЕНЕНИЙ 30

1 ОБЩИЕ СВЕДЕНИЯ

1.1 ОБОЗНАЧЕНИЕ И НАИМЕНОВАНИЕ ПРОГРАММ

Наименование изделия — Компилятор языка МультиОберон со сменными бэкендами «МультиОберон 1.3».

Сокращенное наименование: «МультиОберон».

Программа «МультиОберон» («MultiOberon») зарегистрирована в реестре программ для ЭВМ №2022669920 от 26.10.2022.

Наименование документа 34185873.425510.003.34.М, обозначение по ГОСТ 34.201-89.

МультиОберон является базовым инструментом сборки приложений «СКАДИ» и «СКЗА». «СКАДИ» представляет собой программную платформу средств комплексной автоматизации и диагностики. «СКЗА» представляет собой средства коммуникации зонной архитектуры. Описание языка МультиОберон приведено в документе 34185873.425510.001.35.М «Программная платформа АСУ ТП и диагностики. Описание языка МультиОберон»

1.2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, НЕОБХОДИМОЕ ДЛЯ ФУНКЦИОНИРОВАНИЯ ПРОГРАММЫ

Для функционирования МультиОберона требуется наличие операционной системы семейств Linux, Raspberry PI OS, Windows.

ПО МультиОберон предназначено для работы с операционным программным и аппаратным обеспечением, согласно таблице 1.

Архитектура+ОС	Разрядность	linux	Raspberry Pi OS	windows
X86	32	Binue	--	Binwe
X64	64	Binur	--	Binwr
ArmV71	32	--	Binu4	--
Aarch64	64	--	--	--

Таблица 1 – Поддерживаемые ОС и архитектуры

При выборе доверенных операционных систем рекомендуется использование ОС Astra Linux, начиная с версии 1.6. При выборе Windows рекомендуется использование отечественных аналогов.

1.3 ЯЗЫКИ ПРОГРАММИРОВАНИЯ, НА КОТОРЫХ НАПИСАНА ПРОГРАММА

ПО МультиОберон написана на языке МультиОберон. МультиОберон использует при компиляции программные средства МультиОберон версии 1.3. МультиОберон может быть собран нативными средствами компилятора языка Оберон/Компонентный Паскаль из среды программирования BlackBox.

МультиОберон для конкретных операционных систем использует при компиляции также программные средства МультиОберона. Также имеется возможность кросс-компиляции под другие ОС и архитектуры.

2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

2.1 КЛАССЫ РЕШАЕМЫХ ЗАДАЧ

МультиОберон осуществляет кроссплатформенную сборку приложений, работающую на разных аппаратных архитектурах и операционных системах. МультиОберон это компилятор языка Оберон с тремя различными бэкендами:

- Генератором нативного кода x86 для системы BlackBox;
- Транслятором Ofront в язык C;
- Генератором кода LLVM.

2.2 НАЗНАЧЕНИЕ ПРОГРАММЫ И ОСНОВНЫЕ ФУНКЦИИ

МультиОберон предназначен для конпиляции программного обеспечения диалектов Оберона:

Компонентный Паскаль->Оберон-2->Оберон-07->Оберон.

МультиОберон позволяет устанавливать систему ограничений на требуемые диалекты Оберона. Кроме этого, возможна установка дополнительных ограничений соразмерно требованиям проекта. Компилятор обеспечивает гарантии соответствия кода системе ограничений.

МультиОберон содержит средства трансляции в ANSI C для переноса как компилятора МультиОберона, так и разработанного с его помощью ПО на перспективные отечественные платформы и операционные системы. Для каждой вновь появившейся отечественной программно-аппаратной среды компилятор и ПО на МультиОбероне переносится по стандартной схеме через транслятор в ANSI C.

МультиОберон поддерживает полностью поддерживает концепцию модульного программирования. Каждый программный модуль является единицей хранения, компиляции, загрузки и выполнения. Хранение модулей реализовано в текстовом “.mob” и двоичном “.odc” форматах. Каждому модулю соответствует один файл хранения. Компиляция модулей производится в форматы:

- .ocf – для нативного бэкенда;
- .c – для транслятора в язык C;
- .ll, .bc – для бэкенда LLVM.

Сборка модулей производится в формат “.o” для транслятора в C и LLVM бэкенда.

Динамическая загрузка модулей производится из форматов:

- .ocf – для нативного бэкенда;
- .o – для транслятора в C и LLVM бэкенда.

Линковка исполняемых файлов осуществляется с рекурсивным подключением импортируемых модулей. На основании списков импорта строится дерево проекта, по

которому происходит обход для реализации сборки, линковки или проверки прохождения дерева.

МультиОберон запускается как в консольном режиме, так и из среды BlackBox. Для консольного и графического режимов реализованы варианты компиляторов:

- ombc и OmbCompiler – для нативного бэкенда;
- omfc и OmfCompiler – для транслятора в C;
- omlc и OmlCompiler – для бэкенда LLVM.

Для консольного режима реализованы оболочки, позволяющие осуществлять динамическую загрузку модулей:

- ombsh - для нативного бэкенда;
- omfsh – для транслятора в C;
- omlsh – для бэкенда LLVM.

МультиОберон предназначен для мульти-платформенной разработки. Процедуры одних модулей не зависят от платформо-специфических характеристик, а для других зависимость существует. Эти характеристики включают в себя интерфейсы библиотечных функций операционных систем, различные структуры данных для 32 и 64-битных реализаций, специфических интерфейсов модуля Kernel.

МультиОберон имеет средства для кросс-платформенной разработки. Приложения на платформе с работающим компиляторам могут переноситься на другие целевые платформы. В конфигурационных файлах МультиОберона содержится список настраиваемых опций. Среди указанных опций есть опции с указанием других целевых платформ.

МультиОберон может применяться на всех этапах разработки ПО для критически важных систем как системное программное средство.

2.3 СВЕДЕНИЯ О ФУНКЦИОНАЛЬНЫХ ОГРАНИЧЕНИЯХ НА ПРИМЕНЕНИЕ

Система функциональных ограничений МультиОберон позволяет использовать компилятор МультиОберона для разработки ПО на различных диалектах.

МультиОберон представляет собой масштабируемую технологию на основе систем ограничений с начальной точкой в виде синтаксиса Компонентного Паскаля. Оператор RESTRICT используется для активации/деактивации основных языковых средств Оберона. Подсистема Restrict содержит набор специальных профилей, каждый из которых содержит систему запретов использования ключевых слов на уровне компилятора.

Начальная точка в виде диалекта Компонентного Паскаля имеет пустой профиль. Другие диалекты получаются редуцированием набора ключевых слов и семантических конструкций в профилях, относящихся к данному диалекту.

Концепция масштабирования/демасштабирования диалектов проиллюстрирована на рисунке 1.

Рисунок 1 – Масштабирование, ограничения и функциональность

Переход от начальной точки Оберон-КП к диалекту Оберон-07 осуществляется набором ограничений функциональности в подключаемом профиле Оберон-07. За счет таких введенных ограничений улучшается регулярность данных. Отключение операторов работы с динамической памятью и механизмов сборки мусора улучшает характеристики эргодичности системы. Переход к подмножеству МЭК-880 определяет систему ограничений для соответствия требованиям категории А стандарта МЭК-880. Такие требования обеспечивают постоянство времени прогона ПО. В добавление к уже имеющимся ограничениям включает запрет использования циклов с переменным числом итераций.

Профили МультиОберона представляющие собой файлы определений, включающие запреты:

- NEW, POINTER – использование динамической памяти;
- WHILE, REPEAT, LOOP, EXIT – использование циклов с переменным числом итераций;
- SYSTEM – использование незащищенных указателей;
- PROCEDURE – использование рекурсии и вложенных процедур.

3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1 ОСНОВНЫЕ АЛГОРИТМЫ ПРОГРАММ

МультиОберон основан на архитектуре OP2 с одним фронтендом с со сменными бэкендами, рисунок 2.

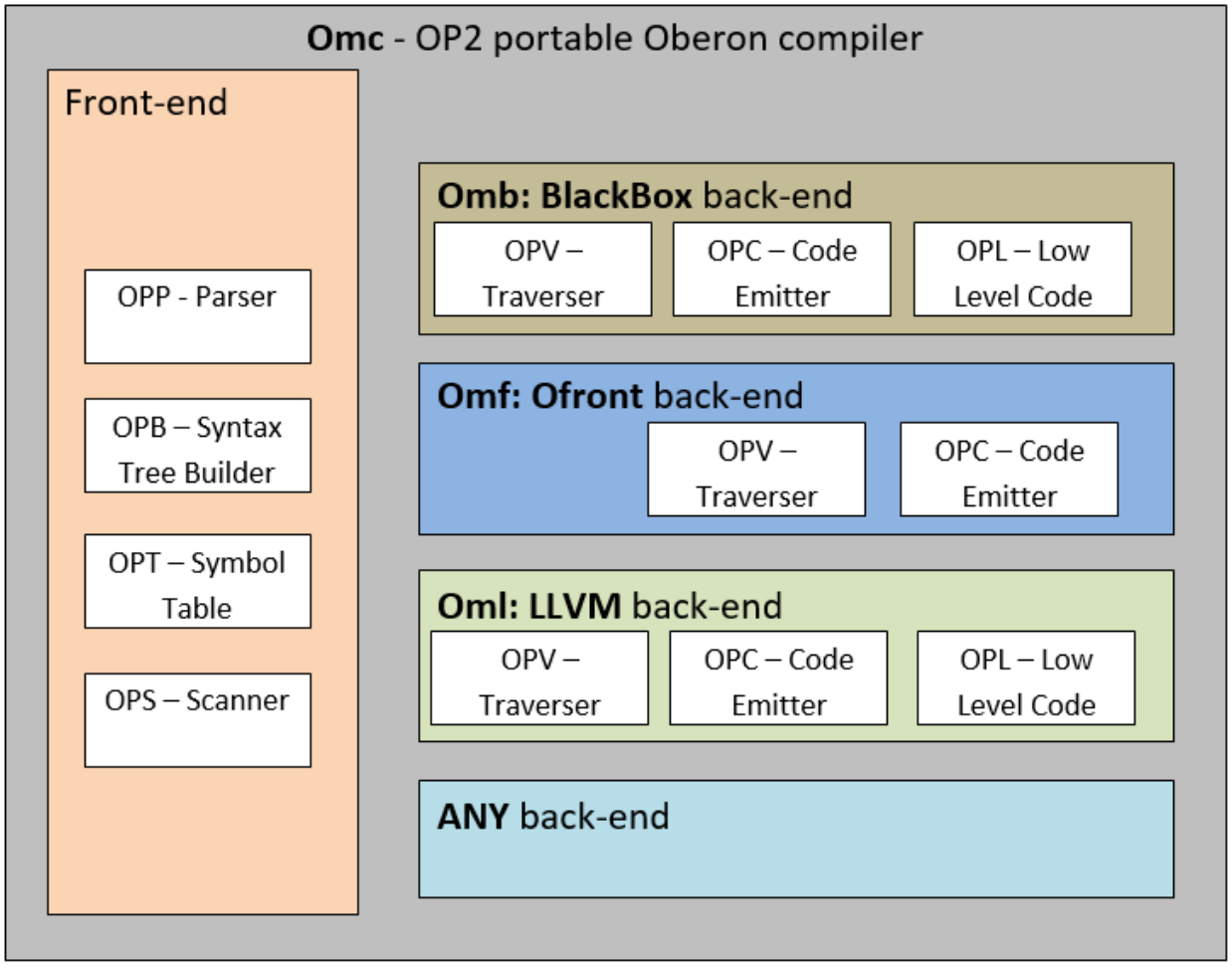

Рисунок 2 – Основные подсистемы МультиОберона

Единая подсистема фронтенда распадается на приведенные на рисунке основные модули:

- OPP парсера входного текста;
- OPB построитель синтаксических деревьев;
- OPT модуль работы с символьными таблицами;
- OPS сканер входной текстовой последовательности.

Помимо основных, в данную подсистему включены модули низкоуровневых функций чтения и преобразования, модули доступа к odc- и mob- файлам, модули работы с ограничениями.

Подсистема фронтенда строит абстрактное синтаксическое дерево, которое впоследствии используется каждым из бэкендов. Абстрактное синтаксическое дерево создается из программного модуля, который является единицей хранения, компиляции, загрузки и выполнения.

Далее включается в работу один выбранный бэкенд, реализующий алгоритмы траверса по дереву OPV-traverse. Траверсе подразумевает нисходящий рекурсивный обход узлов дерева с вызовом процедур кодогенерации в каждом из них. Кодогенерация осуществляется модулем OPC-code emitter с опциональным вызовом модуля низкоуровневых утилит кодогенерации OPL-low level code.

3.2 ИСПОЛЬЗУЕМЫЕ МЕТОДЫ ТРАНСФОРМАЦИИ

МультиОберон использует метод рекурсивного спуска для разбора исходного текста программы и построения абстрактного синтаксического дерева. Сканер и парсер используются при разборе текста, по мере прохождения текста формируется дерево, рисунок 3.

Последовательные операторы (x:=x*y+c; INC(y)) формируют последовательность узлов с командами Nassign. Каждый из узлов содержит либо переменную (Nvar INTEGER x), либо константу (Nconst SHORTINT c 4), либо такой же узел, только вложенный. Вложенный узел также может содержать переменные, константы и узлы.

Инкрементный оператор INC() содержит узел присваивания безымянной константы, равной 1 (SHORTINT NIL 1).

Слева от рисунка приведены операторы, соответствующие данному синтаксическому дереву. Набор операторов не ограничивается операторами присваивания Nassign. В тексте программы могут быть операторы вызова процедур, операторы цикла и т.д.

Кодогенерация, полученная по дереву рекурсивного спуска, формирует неоптимизированный программный код. Присутствует только минимальная оптимизация в виде расчета и подстановки значений констант.

Нативный бэкенд Omb использует непосредственно кодогенерацию из дерева.

Бэкенд Omf транслирует синтаксическое дерево в код на языке C. Такой код может быть далее оптимизирован компиляцией внешней программой, например, gcc с опциями оптимизации –O1 .. –O4.

Бэкенд Omf использует метод оптимизации внешнего компилятора.

Бэкенд Oml компилирует синтаксическое дерево в LLVM байт-код. Такой код может быть далее оптимизирован статическим компилятором llc с опциями оптимизации –O1 .. –O3.

Бэкенд Oml использует метод оптимизации внешнего компилятора.

```
CONST c = 4;
VAR x: INTEGER; y: SHORTINT;
…
x := x * y + c;
INC(y);
…
```

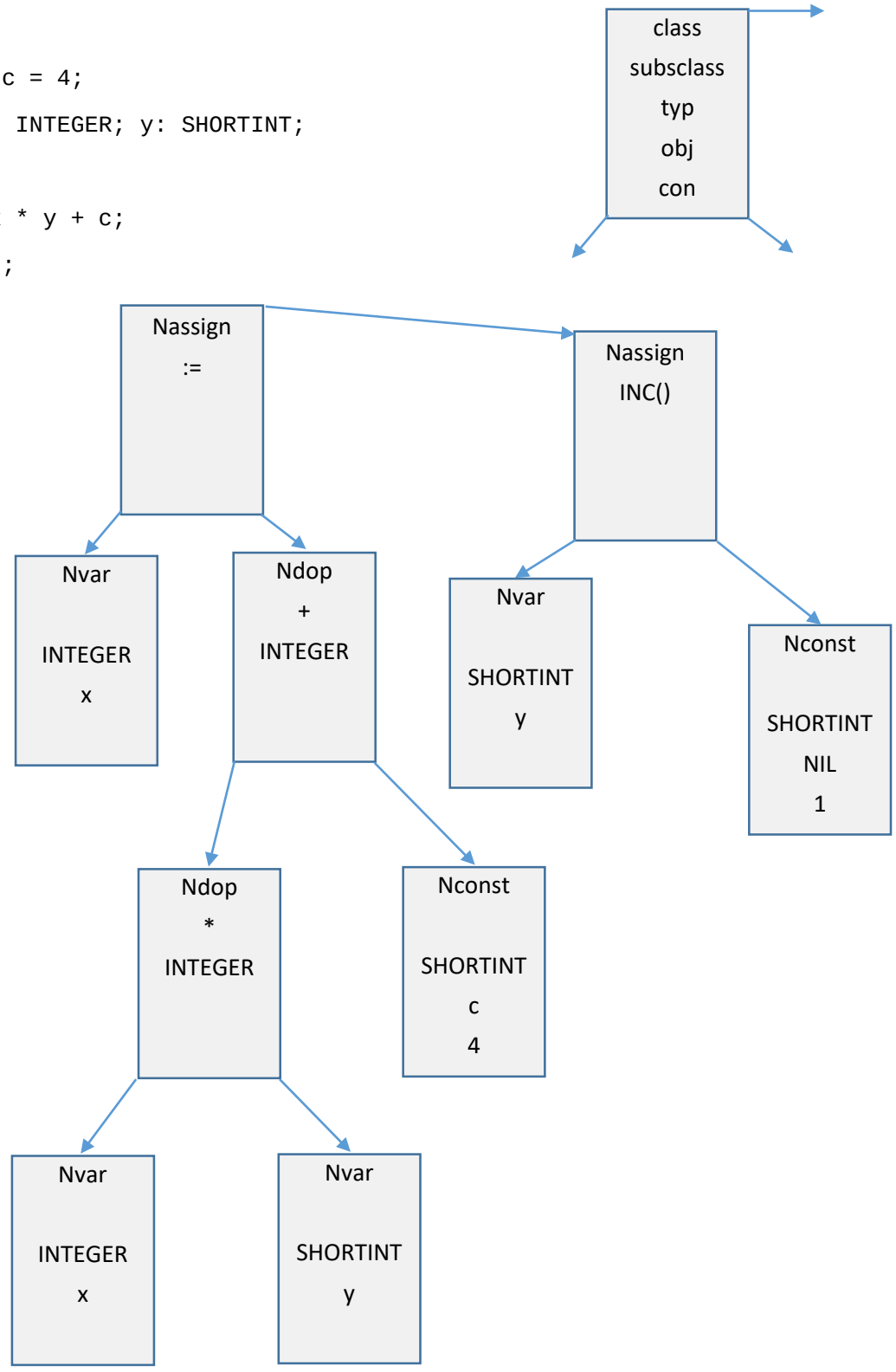

Рисунок 3 – Дерево, полученное методом рекурсивного спуска

3.3 СТРУКТУРА И СОСТАВНЫЕ ЧАСТИ ПО

МультиОберон включает в себя компиляторы и оболочки для всех видов бэкендов, рисунок 4.

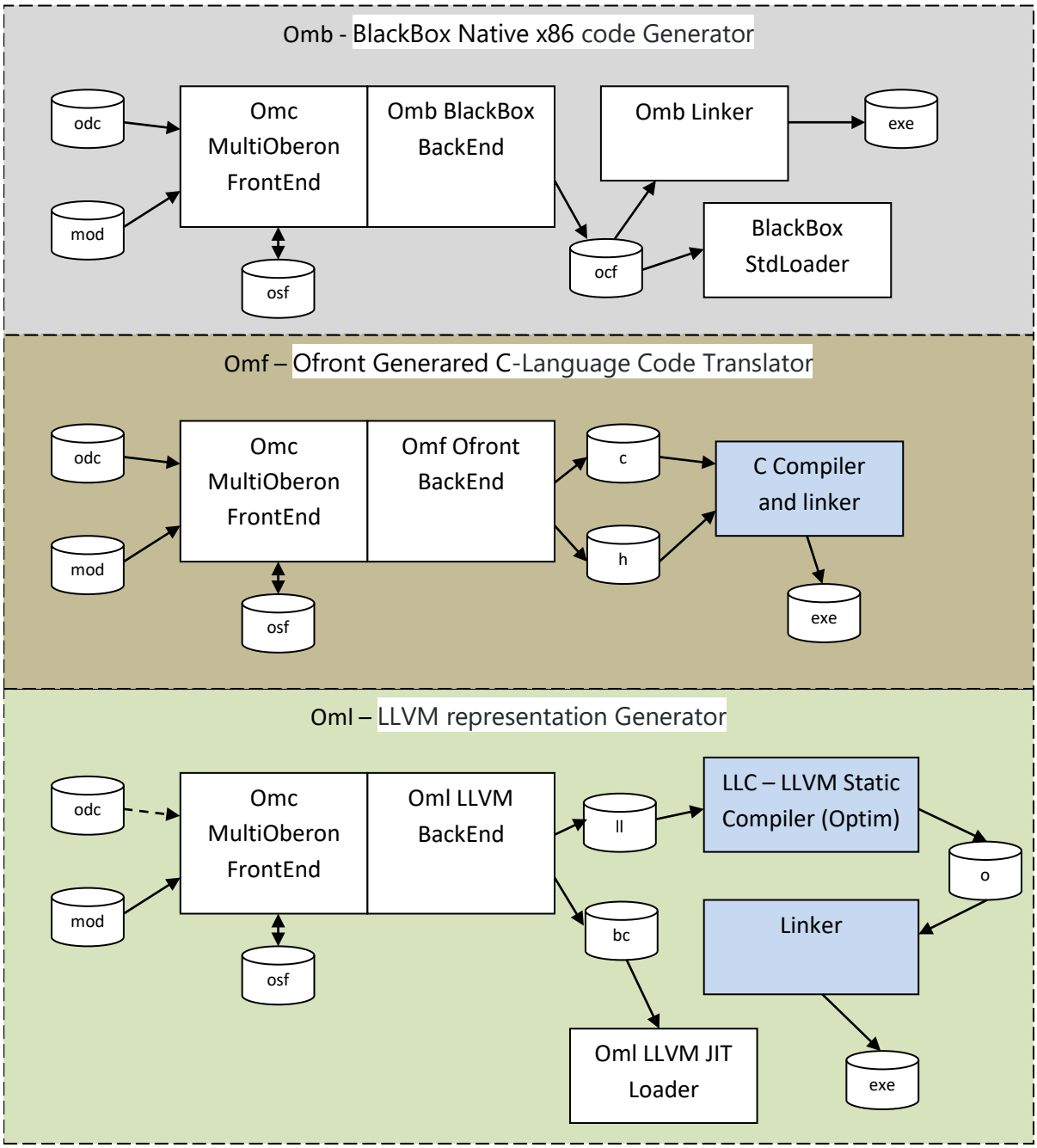

Рисунок 4 – Структура и состав приложений компиляторов

Программное обеспечение фронтенда Omc является общим для всех приложений компилятора. Фронтенд обрабатывает входные файлы *.odc и *.mob и создает деревья и таблицы символов. Далее символьные таблицы модулей сохраняются в виде файлов *.osf для последующего импорта другими модулями.

Программные части бэкендов различаются как структурно, так и в плане взаимодействия с внешней средой.

Бэкенд Omb формирует бинарный код модуля в файлах *.ocf. Такой код предназначен как для последующей загрузки, так и для исполнения. Но исполнение *.ocf возможно только как динамически загружаемый модуль. Для создания статического приложения необходима линковка головного модуля со всеми модулями, им вызываемыми, и далее по рекурсии. Модуль OmbLinker осуществляет необходимые действия по линковке приложения.

Бэкенд Omf формирует текстовые файлы кода *.c и заголовков *.h. Данные файлы предназначены для компиляции и линковке внешней программой. Объектный код *.o может в дальнейшем использоваться для динамической загрузки модуля. В качестве внешней программы может использоваться gcc или clang. Оба внешних приложения имеют средства линковки.

Бэкенд Oml создает байт-код в виде текстового *.ll и бинарного *.bc представления. Использование *.bc для исполнения средствами LLVM JIT Loader приводит к неэффективному использованию процессорного времени. Поэтому основным способом работы является компиляция в объектный код модуля. Эту операцию осуществляет приложение llc – LLVM Static Compiler. Скомпилированный им объектный код *.o может в дальнейшем использоваться для динамической загрузки модуля. Средствами gcc или clang может осуществляться линковка в исполняемые приложения.

Состав и содержание каталогов ПО представлены в приложении 1.

3.4 СВЯЗИ С ДРУГИМИ ПРОГРАММАМИ

Для работы ПО МультиОберон из среды BlackBox необходимо инсталлировать указанную среду. Используется версия 1.7.2 BlackBox (загрузка bbcb-1.7.2~b1.154.tar.gz) по адресу https://blackbox.oberon.org/download

Работа с бэкендом Omf МультиОберон требует установки gcc или clang.

Работа с бэкендом Oml МультиОберон требует установки статического компилятора llc и линкеров в составе gcc или clang.

4 ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Технические средства используются в соответствии со списком поддерживаемых платформ (таблица 1).

Для платформы X86 используются офисные и промышленные 32-битные компьютеры, рабочие станции и сервера с установленными ОС Linux (Ubuntu, Debian, Alt, CentOS) или Windows.

Для платформы X64 используются офисные и промышленные 32-битные компьютеры, рабочие станции и сервера с установленными ОС Linux (Astra, Ubuntu, Debian, Alt, CentOS) или Windows.

Для платформы ArmV71 используются 64-битные технические средства Raspberry Pi 4. ПО МультиОберон устанавливалось на 32-битную Raspberry Pi OS.

Для платформы Aarch64 используются 64-битные технические средства Raspberry Pi 4. ПО МультиОберон устанавливалось на 64-битную OS Ubuntu для Raspberry Pi.

5 ВЫЗОВ И ЗАГРУЗКА

5.1 ДИНАМИЧЕСКИ ЗАГРУЖАЕМЫЕ МОДУЛИ

МультиОберон реализует динамическую загрузку модулей для всех вышеупомянутых бэкендов. Это может быт сделано следующими путями:

```
>Binwe\ombc run OmtestHelloWorld
>Binwe\ombsh OmtestHelloWorld
>Binwe\omlsh OmtestHelloWorld –ext bc
>Binue/ombc run OmtestHelloWorld
>Binue/ombsh OmtestHelloWorld
>Binue/omlsh OmtestHelloWorld –ext bc
```

Первый и второй варианты реализуются для всех платформ всех бэкендов, последний – только для Oml LLVM. Динамически загружаемый модуль должен быть в подготовленной бинарной форме согласно таблице 2.

	Windows	Unix	Unix on Arm
Omb	.ocf BlackBox	.ocf BlackBox	--
Omf	.o COFF-format	.o ELF-format	.o ELF-format
Oml	.o COFF-format	.o ELF-format	.o ELF-format
Oml –ext bc	.bc LLVM	.bc LLVM	.bc LLVM

Таблица 2 – Форматы файлов для динамической загрузки

Файлы .ocf и .bc files создаются при компиляции.
Объектные файлы .o создаются при сборке внешним линкером.

Использование .bc файлов LLVM активирует компилятор LLVM JIT при наличии опции –ext bc. Использование JIT избыточному потреблению процессорного времени и представляется непрактичным и слишком ресурсо-емким.

Для реализации всех упомянутых свойств, базовый модуль загрузчика Baseloader реализован в подсистеме каталога System. Модули, расширяющие тип загрузчика OmcOcfLoader, OmcObjLoader__Coff, OmcObjLoader__Elf, и OmlBcLoader реализуют специфическую функциональность загрузки ocf, coff, elf и bc.

5.2 ТОЧКИ ВХОДА В ПРОГРАММУ

МультиОберон использует специальный модуль Runner с вариациями для различных сред. Процедура Runner.SetRun регистрирует функцию MAIN которая должна быть вызвана после загрузки оболочки.

```
MODULE OmtestMkTraps;
	IMPORT Runner, OLog, SYSTEM;
	PROCEDURE MAIN*;
		VAR ar: Runner.ArgsReader; str: Runner.SName;
	BEGIN
		IF ~ar.StringOpt("-trap", str) THEN
			OLog.String("usage: "); OLog.SString(Runner.argV0);
			OLog.String(" -trap"); OLog.Ln;
			OLog.Tab; OLog.String("where -trap is as following:");
			OLog.Ln;
			OLog.Tab; OLog.String("a - assert"); OLog.Ln;
			OLog.Tab; OLog.String("h - halt"); OLog.Ln;
```

```
                OLog.Tab; OLog.String("z - zero divide"); OLog.Ln;
                OLog.Tab; OLog.String("p - nil pointer dereference");
                OLog.Ln;
          ELSE
                RunOpt(str);
          END;
    END MAIN;

BEGIN
    Runner.SetRun(MAIN)
END OmtestMkTraps.
```

Runner также содержит разбор опций командной строки процедурами Runner.ArgsReader.StringOpt(), Runner.ArgsReader.IntOpt().

Константы модуля Runner определяют величины, специфические для различных платформ:

Runner.RUN_TIME = “OMB” | “OMF” | “OML” для BlackBox, Ofront, LLVM сред.
Runner.OS_NAME = “Windows” | “Unix”
Runner.BIN_BITS = 64 | 32
Runner.KERNEL_VERSION = 16 | 17 | 18 для BlackBox Kernel
Runner.DESC_MACH = “X86” | “X64” | “ARM32” | “ARM64”

6 ВХОДНЫЕ ДАННЫЕ

6.1 ДЖЕНЕРИКИ И ПРЕПРОЦЕССОР

Дженерики как понятие не включены в компилятор МультиОберона. Равно как и не планируется их включение в будущем. Однако, минимальная утилита конвертации .odc включена в систему Omc.

Модуль OmcPrep представляет собой текстовый препроцессор для конвертации из дженериков в специфические модули.

По конвенции МультиОберона, дженерики расположены в каталоге GMod, специфические модули расположены в каталоге Mod. Специфические модули (пример в System/Docu/Quick-Start.odc) генерятся вызовом препроцессора из модулей дженериков. Например, команда генерации Runner__fwr17 из GMod/Runner выглядит следующим образом:

```
^Q OmcPrep.ToOdcFileList('OMF WIN V64 BB17', 'System/Mod')" @Omc/Mod/Defs.odc
System/GMod/Runner.odc:Runner__fwr17
```

Сначала устанавливаются параметры окружения: OMF, WIN, V64, and BB17. Далее загружается файл определений Defs со специфическими установками, как-то @ADDR=LONGINT (это 64-бит, V64 установлено как параметр окружения). Далее GMod/Runner трансформируется в Runner__fwr17.

```
#IF @BB
            SP = 4; DLT_STACK = 256;
            RUN_TIME* = "OMB";
#ELSIF @OMF
            RUN_TIME* = "OMF";
#ELSIF @OML
            RUN_TIME* = "OML";
#END
            SysTrapProc = PROCEDURE (n: INTEGER; stpa: @ADDR);
```

Результатом трансформации для модуля Runner__fwr17 будут:

```
      RUN_TIME* = "OMF";
      SysTrapProc = PROCEDURE (n: INTEGER; stpa: LONGINT);
```

Файл определений Defs содержит специальные настройки для всех необходимых платформ.

Используются только макро-команды условной компиляции и механизм подстановки имен.

6.2 ОПЕРАЦИИ ОГРАНИЧЕНИЯ

МультиОберон это масштабируемая технология на основе систем ограничений с начальной точкой в виде синтаксиса Компонентного Паскаля. Оператор RESTRICT используется для активации/деактивации основных языковых средств Оберона. Подсистема Restrict содержит набор специальных профилей. Модуль RestrictAdrint просто включает ADRINT как целое размерностью адреса:

```
RESTRICT +ADRINT*;
```

Символ ‘*’ означает, что данное ограничение экспортируется в модуль, который непосредственно импортирует RestrictAdrint.

Профиль RestrictOberon07 использует более сложные правила, например:

```
RESTRICT -CLOSE*, -EXIT*, -LOOP*, -OUT*, -WITH*,
      -PROCEDURE(PROCEDURE)*, (* Recursion *)
      -BEGIN(PROCEDURE)*,     (* Nested Procedures *)
      -RETURN(PROCEDURE)*;    (* Single Return in the End only *)
```

Тест OmtestOmcRestrict_test демонстрирует использование ADRINT. Переменная типа ADRINT может сохранять адресс в качестве целочисленной величины.

```
IMPORT Api, RestrictAdrint;
PROCEDURE Xxx;
      VAR ai: ADRINT;
BEGIN
      ai := SYSTEM.VAL(ADRINT, Api.TestGetAdr());
END Xxx;
```

Тест OmtestOmcRestrict_test компилируется и выполняется:

```
>Binwe\ombc co -odc OmtestOmcRestrict_test
>Binue/ombc co -odc OmtestOmcRestrict_test
>Binwe\ombc test –pl 2 OmtestOmcRestrict_test
>Binue/ombc test –pl 2 OmtestOmcRestrict_test
[ALL] ========== Total 14 tests, 0 bad, result= 100.0%
```

Дополнительная функциональность RESTRICT+ реализована в текущей версии только для ADRINT.

7 ВЫХОДНЫЕ ДАННЫЕ

7.1 ПРИМЕРЫ ТЕКСТОВОЙ ГЕНЕРАЦИИ ВЫХОДНЫХ ФАЙЛОВ

МультиОберон осуществляет трансформацию программных файлов из текстового в иные различного рода форматы. Приложение 2 иллюстрирует пример текстового входного файла.

Приложение 3 демонстрирует сгенеренный для данного файла программный код на языке C с помощью бэкенда Omf.

Приложение 4 демонстрирует сгенеренный для данного файла программный код LLVM, полученный посредством бэкенда Oml.

7.2 РАЗРАБОТКА ПЛАТФОРМО-ЗАВИСИМЫХ МОДУЛЕЙ

МультиОберон предназначен для мульти-платформенной разработки. Процедуры одних модулей не зависят от платформо-специфических характеристик, а для других зависимость существует. Эти характеристики включают в себя интерфейсы библиотечных функций операционных систем, различные структуры данных для 32 и 64-битных реализаций, специфических интерфейсов модуля Kernel.

Каждый исходник модуля может состоять из специфических версий, помеченных ДВУМЯ ПОДЧЕРКИВАНИЯМИ как [module]__[specific]. Имена модулей в МультиОбероне не могут содержать 2 символа подчеркивания внутри или один на конце, иначе возникает ошибка “string expected”. Однако, имена файлов могут иметь подчеркивания и специфические расширения для модулей. Более того, наличие подчеркивания в имени файла означает платформо-зависимую реализацию модуля. Например, есть две абсолютно разные реализации модуля OmcObjLoader:

- OmcObjLoader__Coff – для Windows формата COFF;
- OmcObjLoader__Elf – для Unix формата ELF.

Так что OmcObjLoader__Coff компилируется в OmcObjLoader.o под Windows в одном скрипте и OmcObjLoader__Elf компилируется в OmcObjLoader.o под Unix в другом скрипте.

Очевидно, платформо-зависимые модули имеют отличающиеся интерфейсы. Компиляция платформо-независимого модуля:

```
> Binwe\ombc co -odc OmcDiscomp
omb:compiling c:\suok5\dsu/Omc/Mod/Discomp.odc
new symbol file >c:\suok5\dsu/Omc/Code/Discomp.ocf code=3652 glob=8716
```

Располагаются бинарные файлы платформо-зависимых модулей для BlackBox в Omc/Sym и Omc/Code для кода.

Компиляция OmcObjLoader__Coff означает платформо-зависимость:

```
> Binwe\ombc co -odc OmcObjLoader__Coff
omb:compiling c:\suok5\dsu/Omc/Mod/ObjLoader__Coff.odc
new symbol file >c:\suok5\dsu/Omc/Cbwe/ObjLoader.ocf code=18480 glob=32
```

Платформо-зависимые для BlackBox будут в Omc/Sbwe и Omc/Cbwe для кода.

Но даже независимые модули Ofront и LLVM разделяются по каталогам:

```
> Binwe\omfc co -odc OmcDiscomp
omf:compiling c:\suok5\dsu/Omc/Mod/Discomp.odc
>c:\suok5\dsu/Omc/Cfwe/OmcDiscomp .c=26490 .h=3555
```

Платформо-независимые для Ofront будут в Omc/Sfwe и Omc/Cfwe для кода.

Большая часть платформо-зависимых модулей расположена в System и Host. Они компилируются на этапе инсталляции отдельно от пользовательского процесса разработки.

Соглашение по именам использует следующие буквы в МультиОбероне:

- b-BlackBox, f-Ofront, l-LLVM;
- w-Windows, u-Unix;
- e-X86, r-X64;
- 16-BlackBox 1.6, 17-BlackBox1.7.

Модуль Runner требует 19 платформо-зависимых реализаций кода. Примеры подобных соглашений иллюстрирует таблица 3.

	Omb	Omf32	Omf64	Oml32	Oml64
BlackBox17Win	Runner__bwe17	Runner__fwe17	Runner__fwr17	Runner__lwe17	Runner__lwr17
BlackBox16Win	Runner__bwe16	Runner__fwe16	Runner__fwr16	Runner__lwe16	Runner__lwr16
BlackBox17Unix	Runner__bue17	Runner__fue17	Runner__fur17	Runner__lue17	Runner__lur17
17 ArmV71	--	Runner__fu4	--	Runner__lu4	--
17 Aarc64	--	--	Runner__fu8	--	Runner__lu8

Таблица 3 – Платформо-зависимые коды Runner

На усмотрение пользователя остается решение – разрабатывать каждый платформо-зависимый модуль в отдельности, либо использовать дженерики и препроцессор, описанные в предыдущих секциях. В приведенном в таблице примере все 19 платформо-зависимых модуля были сгенерены с помощью препроцессора дженериков.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1 – СОСТАВ КАТАЛОГОВ МУЛЬТИОБЕРОН

МультиОберон состоит из исполняемых файлов приложений и списка программных подсистем. Каталоги бинарных исполняемых файлов соответствуют правилу Bin[os][arch], таблица 1-1.

Архитектура+ОС	windows	unix (linux, …)
X86	Binwe	Binue
X64	Binwr	Binur
ArmV71	--	Binu4
Aarch64	--	Binu8

Таблица 1-1 – Именование каталогов бинарных исполняемых файлов

Бинарные приложения состоят из компиляторов и оболочек (shells). Компиляторы МультиОберона включают в себя полную поддержку функций компиляции и следуют правилу om[backend]c. Оболочки МультиОберона поддерживают только динамическую загрузку и исполнение, файлы соответствуют правилу om[backend]sh. Имена бинарных приложений приведены в таблице 1-2.

	BlackBox	Ofront	LLVM
компилятор	ombc	omfc	omlc
оболочка	ombsh	omfsh	omlsh

Таблица 1-2 – Имена бинарных приложений

Программные подсистемы структурируются по по правилам подсистем BlackBox.

Бэкенд следует правилу Om[backend]. Третья буква означает следующие варианты:

- Omc это компилятор и консоль МультиОберона;
- Omb является реализацией компилятора для бэкенда BlackBox;
- Omf является реализацией компилятора для бэкенда Ofront;
- Oml является реализацией компилятора для бэкенда LLVM. Использует библиотеку, подготовленную из LLVM 5.0.

Каждая подсистема имеет каталог Mod для исходников, Docu для документов, каталоги с кодовыми и символьными файлами.

Каталоги кодовых файлов следуют правилу C[backend][os][arch], таблица 1-3.

Архитектура+ОС	windows	Unix (Linux, …)
X86 BlackBox portable	Code	Code
X86 Omb-specific	Cbwe	Cbue
X86 OFront	Cfwe	Cfue
X64 OFront	Cfwr	Cfur
X86 LLVM	Clwe	Clue
X64 LLVM	Clwr	Clur
ArmV71 OFront	--	Cfu4
Aarch64 OFront	--	Cfu8
ArmV71 LLVM	--	Clu4
Aarch64 LLVM	--	Clu8

Таблица 1-3 - Каталоги кодовых файлов

Программные подсистемы структурируются по правилам подсистем BlackBox. Символьные файлы и файлы использования расположены в символьных каталогах. Каталоги символьных файлов следуют правилу S[backend][os][arch], таблица 1-4.

Архитектура+ОС	windows	Unix (Linux, …)
X86 BlackBox portable	Sym	Sym
X86 Omb-specific	Sbwe	Sbue
X86 OFront	Sfwe	Sfue
X64 OFront	Sfwr	Sfur
X86 LLVM	Slwe	Slue
X64 LLVM	Slwr	Slur
ArmV71 OFront	--	Sfu4
Aarch64 OFront	--	Sfu8
ArmV71 LLVM	--	Slu4
Aarch64 LLVM	--	Slu8

Таблица 1-4 - Каталоги символьных файлов

ПРИЛОЖЕНИЕ 2 – ПРИМЕР HELLO WORLD

Ниже приведен исходный код программы OmtestHelloWorld.

```
MODULE OmtestHelloWorld;
(**
	project	= "MultiOberon Compiler"
	contributors	= "Dmitry V.Dagaev"
	license	=  "LGPL version 3"
	narrative = "Test for Hello, World and other main prints"
**)

	IMPORT Runner, OLog, Kernel;

	PROCEDURE MAIN*;
	BEGIN
		OLog.String("Hello, World");
		OLog.Ln;
	END MAIN;

BEGIN
	Runner.SetRun(MAIN)
END OmtestHelloWorld.
```

ПРИЛОЖЕНИЕ 3 – СГЕНЕРЕННЫЙ С-КОД ДЛЯ HELLO WORLD

Ниже приведен сгенеренный код для OmtestHelloWorld. Помимо собственно кода в тексте программы содержится информация рантайма о типах и процедурах.

```
/*  Omf-1.0 k -directories directories /all_sys_val */
#include "SYSTEM.h"
#include "Runner.h"
#include "Kernel.h"
#include "OLog.h"
#include "HostConLog.h"

export void OmtestHelloWorld__MAIN (void);

export void OmtestHelloWorld___reg();
export void OmtestHelloWorld___body();
export struct SYSTEM__MODDESC OmtestHelloWorld___desc;

void OmtestHelloWorld__MAIN (void)
{
      __ENTER("OmtestHelloWorld.MAIN");
      (*OLog__String)((_CHAR*)L"Hellow, World", 14);
      (*OLog__Ln)();
      __EXIT;
}

export void OmtestHelloWorld___reg()
{
      __BEGREG(OmtestHelloWorld___desc);
      Runner___reg();
      OLog___reg();
      HostConLog___reg();
      __REGMOD(OmtestHelloWorld___desc);
      __ENDREG;
}

export void OmtestHelloWorld___body()
{
      __BEGBODY(OmtestHelloWorld___desc);
      Runner___body();
      OLog___body();
      HostConLog___body();
      Runner__SetRun(OmtestHelloWorld__MAIN);
      __ENDBODY;
}

export int main(int argc, char **argv)
{
      __BEGMAIN(argc,argv);
      OmtestHelloWorld___reg();
      OmtestHelloWorld___body();
      return 0;
}

static ADRINT OmtestHelloWorld__MAIN__sig[] = {
      0,
      0,
};
static SYSTEM__MODDESC *OmtestHelloWorld___imp[] = {
      &OLog___desc,
      &Runner___desc,
```

```
};
static ADRINT OmtestHelloWorld___exp[] = {
      1,
      0x5814f4d6, (ADRINT)OmtestHelloWorld__MAIN, 1<<8 | 0x44,
(ADRINT)OmtestHelloWorld__MAIN__sig,
};
static char OmtestHelloWorld___names[] = {
      0,
      'M','A','I','N',0,
};
export char OmtestHelloWorld___inames[] = {
      'O','m','t','e','s','t','H','e','l','l','o','W','o','r','l','d',0,
      'R','u','n','n','e','r',0,
      'K','e','r','n','e','l',0,
      'O','L','o','g',0,
      0,
};
static ADRINT OmtestHelloWorld___ptrs[] = {
      0
};
export int OmtestHelloWorld___oftag = 0X6F4F4346;
struct SYSTEM__MODDESC OmtestHelloWorld___desc = {
      0, 13, 0, /* next, opts, refcnt */
      {2023, 2, 10, 15, 13, 3}, /* compTime */
      {0, 0, 0, 0, 0, 0}, /* loadTime */
      0, /* no body */
      0,
      2, /* nofimps */
      0, /* nofptrs */
      0, 0, 0, 0, /* csize..code */
      0, 0, 0, 0, /* data..varBase */
      OmtestHelloWorld___names,
      OmtestHelloWorld___ptrs,
      OmtestHelloWorld___imp,
      (SYSTEM__DIRECTORY*)OmtestHelloWorld___exp,
      "OmtestHelloWorld"
};
```

ПРИЛОЖЕНИЕ 4 – ТЕКСТОВЫЙ LLVM-КОД ДЛЯ HELLO WORLD

Ниже приведен текст LLVM-кода для для OmtestHelloWorld.

```
; ModuleID = 'OmtestHelloWorld'
source_filename = "OmtestHelloWorld"

%SYSTEM__MODDESC = type { %SYSTEM__MODDESC*, i32, i32, [6 x i16], [6 x i16],
void ()*, void ()*, i32, i32, i32, i32, i32, i32, i32, i32, i32, i32, [0 x
i8]*, [1 x i32]*, [4 x %SYSTEM__MODDESC*]*, %SYSTEM__DIRECTORY*, [256 x i8] }
%SYSTEM__DIRECTORY = type { i32, [1 x %SYSTEM___OBJDESC] }
%SYSTEM___OBJDESC = type { i32, i32, i32, i32 }
%SYSTEM__DLINK = type { %SYSTEM__DLINK*, %SYSTEM__MODDESC*, [0 x i8]* }

@OmtestHelloWorld___imp = private global [4 x %SYSTEM__MODDESC*]
[%SYSTEM__MODDESC* @Runner___desc, %SYSTEM__MODDESC* @Kernel___desc,
%SYSTEM__MODDESC* @OLog___desc, %SYSTEM__MODDESC* null]
@Runner___desc = external global %SYSTEM__MODDESC
@Kernel___desc = external global %SYSTEM__MODDESC
@OLog___desc = external global %SYSTEM__MODDESC
@OmtestHelloWorld___inames = global [37 x i8]
c"OmtestHelloWorld\00Runner\00Kernel\00OLog\00\00"
@OmtestHelloWorld___ptrs = private global [1 x i32] zeroinitializer
@OmtestHelloWorld___exp = private global %SYSTEM__DIRECTORY { i32 1, [1 x
%SYSTEM___OBJDESC] [%SYSTEM___OBJDESC { i32 1477768406, i32 ptrtoint (void
()* @OmtestHelloWorld__MAIN to i32), i32 2116, i32 ptrtoint ([2 x i32]*
@OmtestHelloWorld__MAIN__sig to i32) }] }
@OmtestHelloWorld___desc = global %SYSTEM__MODDESC { %SYSTEM__MODDESC* null,
i32 3, i32 0, [6 x i16] [i16 2023, i16 2, i16 16, i16 16, i16 10, i16 18], [6
x i16] zeroinitializer, void ()* @OmtestHelloWorld___body, void ()* null, i32
3, i32 0, i32 0, i32 0, i32 0, i32 0, i32 0, i32 0, i32 0, i32 0, [0 x i8]*
bitcast ([25 x i8]* @OmtestHelloWorld___names to [0 x i8]*), [1 x i32]*
@OmtestHelloWorld___ptrs, [4 x %SYSTEM__MODDESC*]* @OmtestHelloWorld___imp,
%SYSTEM__DIRECTORY* @OmtestHelloWorld___exp, [256 x i8]
c"OmtestHelloWorld\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\0
0\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\
00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00" }
@Kernel__dLink = external global %SYSTEM__DLINK*
@OmtestHelloWorld___oftag = global i32 1867465542
@n_OmtestHelloWorld__MAIN = private global [5 x i8] c"MAIN\00"
@OLog__StringProc__desc = external global [21 x i32]
@OLog__String = external global void ([0 x i16]*, i32)*
@OLog__Proc__desc = external global [21 x i32]
@OLog__Ln = external global void ()*
@n_OmtestHelloWorld___reg = private global [5 x i8] c"_reg\00"
@OmtestHelloWorld__MAIN__sig = private global [2 x i32] zeroinitializer
@n_OmtestHelloWorld___body = private global [6 x i8] c"_body\00"
@Runner__RunProc__desc = external global [21 x i32]
@n_OmtestHelloWorld___main = private global [6 x i8] c"_main\00"
@OmtestHelloWorld___names = private global [25 x i8]
c"\00Kernel\00MAIN\00OLog\00Runner\00"

define void @OmtestHelloWorld___reg() {
entry:
```

```
  %dlink = alloca %SYSTEM__DLINK
  %next = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 0
  %lda = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  store %SYSTEM__DLINK* %lda, %SYSTEM__DLINK** %next
  %mod = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 1
  store %SYSTEM__MODDESC* @OmtestHelloWorld___desc, %SYSTEM__MODDESC** %mod
  %procname = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink,
i32 0, i32 2
  store [0 x i8]* bitcast ([5 x i8]* @n_OmtestHelloWorld___reg to [0 x i8]*),
[0 x i8]** %procname
  store %SYSTEM__DLINK* %dlink, %SYSTEM__DLINK** @Kernel__dLink
  %lda1 = load i32, i32* getelementptr inbounds (%SYSTEM__MODDESC,
%SYSTEM__MODDESC* @OmtestHelloWorld___desc, i32 0, i32 1)
  %AND = and i32 %lda1, 262144
  %ICMP = icmp ne i32 %AND, 0
  br i1 %ICMP, label %then, label %merge

then:                                             ; preds = %entry
  %lda2 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next3 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda2, i32
0, i32 0
  %lda4 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next3
  store %SYSTEM__DLINK* %lda4, %SYSTEM__DLINK** @Kernel__dLink
  ret void

merge:                                            ; preds = %entry
  %OR = or i32 %lda1, 262144
  store i32 %OR, i32* getelementptr inbounds (%SYSTEM__MODDESC,
%SYSTEM__MODDESC* @OmtestHelloWorld___desc, i32 0, i32 1)
  call void @Kernel___reg()
  call void @Runner___reg()
  call void @OLog___reg()
  call void @HostConLog___reg()
  call void @Kernel__RegisterStaticMod(%SYSTEM__MODDESC*
@OmtestHelloWorld___desc)
  %lda5 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next6 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda5, i32
0, i32 0
  %lda7 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next6
  store %SYSTEM__DLINK* %lda7, %SYSTEM__DLINK** @Kernel__dLink
  ret void
}

define void @OmtestHelloWorld___body() {
entry:
  %dlink = alloca %SYSTEM__DLINK
  %next = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 0
  %lda = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  store %SYSTEM__DLINK* %lda, %SYSTEM__DLINK** %next
  %mod = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 1
  store %SYSTEM__MODDESC* @OmtestHelloWorld___desc, %SYSTEM__MODDESC** %mod
  %procname = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink,
i32 0, i32 2
  store [0 x i8]* bitcast ([6 x i8]* @n_OmtestHelloWorld___body to [0 x
i8]*), [0 x i8]** %procname
  store %SYSTEM__DLINK* %dlink, %SYSTEM__DLINK** @Kernel__dLink
```

```
  %lda1 = load i32, i32* getelementptr inbounds (%SYSTEM__MODDESC,
%SYSTEM__MODDESC* @OmtestHelloWorld___desc, i32 0, i32 1)
  %AND = and i32 %lda1, 65536
  %ICMP = icmp ne i32 %AND, 0
  br i1 %ICMP, label %then, label %merge

then:                                             ; preds = %entry
  %lda2 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next3 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda2, i32
0, i32 0
  %lda4 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next3
  store %SYSTEM__DLINK* %lda4, %SYSTEM__DLINK** @Kernel__dLink
  ret void

merge:                                            ; preds = %entry
  %OR = or i32 %lda1, 65536
  store i32 %OR, i32* getelementptr inbounds (%SYSTEM__MODDESC,
%SYSTEM__MODDESC* @OmtestHelloWorld___desc, i32 0, i32 1)
  call void @Kernel___body()
  call void @Runner___body()
  call void @OLog___body()
  call void @HostConLog___body()
  call void @Runner__SetRun(void ()* @OmtestHelloWorld__MAIN)
  %lda5 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next6 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda5, i32
0, i32 0
  %lda7 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next6
  store %SYSTEM__DLINK* %lda7, %SYSTEM__DLINK** @Kernel__dLink
  ret void
}

define void @OmtestHelloWorld__MAIN() {
entry:
  %dlink = alloca %SYSTEM__DLINK
  %next = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 0
  %lda = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  store %SYSTEM__DLINK* %lda, %SYSTEM__DLINK** %next
  %mod = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 1
  store %SYSTEM__MODDESC* @OmtestHelloWorld___desc, %SYSTEM__MODDESC** %mod
  %procname = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink,
i32 0, i32 2
  store [0 x i8]* bitcast ([5 x i8]* @n_OmtestHelloWorld__MAIN to [0 x i8]*),
[0 x i8]** %procname
  store %SYSTEM__DLINK* %dlink, %SYSTEM__DLINK** @Kernel__dLink
  %atmp = alloca [27 x i8]
  store [27 x i8] c"H\00e\00l\00l\00o\00,\00
\00W\00o\00r\00l\00d\00\00\00\00", [27 x i8]* %atmp
  %lda1 = load void ([0 x i16]*, i32)*, void ([0 x i16]*, i32)**
@OLog__String
  %PCAST = bitcast [27 x i8]* %atmp to [0 x i16]*
  call void %lda1([0 x i16]* %PCAST, i32 13)
  %lda2 = load void ()*, void ()** @OLog__Ln
  call void %lda2()
  %lda3 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next4 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda3, i32
0, i32 0
  %lda5 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next4
  store %SYSTEM__DLINK* %lda5, %SYSTEM__DLINK** @Kernel__dLink
```

```
  ret void
}

declare void @Kernel___reg()

declare void @Runner___reg()

declare void @OLog___reg()

declare void @HostConLog___reg()

declare void @Kernel__RegisterStaticMod(%SYSTEM__MODDESC*)

declare void @Kernel___body()

declare void @Runner___body()

declare void @OLog___body()

declare void @HostConLog___body()

declare void @Runner__SetRun(void ()*)

define i32 @main(i32, i8**) {
entry:
  %dlink = alloca %SYSTEM__DLINK
  %next = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 0
  %lda = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  store %SYSTEM__DLINK* %lda, %SYSTEM__DLINK** %next
  %mod = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink, i32
0, i32 1
  store %SYSTEM__MODDESC* @OmtestHelloWorld___desc, %SYSTEM__MODDESC** %mod
  %procname = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %dlink,
i32 0, i32 2
  store [0 x i8]* bitcast ([6 x i8]* @n_OmtestHelloWorld___main to [0 x
i8]*), [0 x i8]** %procname
  store %SYSTEM__DLINK* %dlink, %SYSTEM__DLINK** @Kernel__dLink
  call void @Kernel__Main(i32 0, i32 %0, i8** %1)
  call void @OmtestHelloWorld___reg()
  call void @OmtestHelloWorld___body()
  %lda1 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** @Kernel__dLink
  %next2 = getelementptr inbounds %SYSTEM__DLINK, %SYSTEM__DLINK* %lda1, i32
0, i32 0
  %lda3 = load %SYSTEM__DLINK*, %SYSTEM__DLINK** %next2
  store %SYSTEM__DLINK* %lda3, %SYSTEM__DLINK** @Kernel__dLink
  ret i32 0
}

declare void @Kernel__Main(i32, i32, i8**)
```

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

АЭС - Атомная Электростанция

КСА - Комплекс средств автоматизации

ОС - Операционная система

ПО - Программное обеспечение

LLVM - Low Level Virtual Machine – Виртуальная низкоуровневая машина

СКАДИ - Средства комплексной автоматизации и диагностики

ЧМИ - Человеко-машинный интерфейс

ICMP - Internet Control Message Protocol

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

ЛИСТ ИЗМЕНЕНИЙ

Изм.	Номера листов				Всего листов в изме-ненном документе	Номер документа, утверждаю-щего изменение	Подп.	Дата
	измененных	замененных	новых	аннулированных				