ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ

«СКАДИ»

(ООО «СКАДИ»)

УТВЕРЖДАЮ

КОМПИЛЯТОР ЯЗЫКА МУЛЬТИОБЕРОН СО СМЕННЫМИ БЭКЕНДАМИ

Руководство оператора

34185873.425510.003.34.М

(На 28 листах)

2023

РАЗРАБОТАЛ

СОДЕРЖАНИЕ

1 НАЗНАЧЕНИЕ ПРОГРАММЫ

1.1 СВЕДЕНИЯ О НАЗНАЧЕНИИ

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»

МультиОберон предназначен для кроссплатформенной сборки программного обеспечения на диалектах языка Оберон. Программа применяется для сборки систем, работающих в режиме онлайн с повышенными требованиями в части функциональной безопасности.

1.2 ОСНОВНЫЕ ФУНКЦИИ

МультиОберон был предложен для реализации механизма редуцирования диалектов Оберона к минималистическим подмножествам:

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

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. Профили МультиОберона включают 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.

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 позволяет осуществлять 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.

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 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

2.1 ОБЩИЕ ТРЕБОВАНИЯ

МультиОберон поддерживает следующие ОС и платформы:

- Windows X86;
- Windows X64;
- Linux X86;
- Linux X64;
- Raspberry Pi OS ArmV71;
- Ubuntu Linux Aarch64 (64-bit Raspberry Pi).

Для использования в промышленных применениях рекомендуется использование ОС Astra Linux, начиная с версии 1.6.

2.2 УСЛОВИЯ КОНСОЛЬНОГО РЕЖИМА

МультиОберон должен быть корректно инсталлирован. Стандартный каталог для Linux-версий: /usr/local/bin/multioberon.

Пути к исполняемым файлам устанавливаются из переменных окружения, либо записываются в файлы e_path.dat (r_path.dat для 64-бит) и считываются следующим образом.

```
PATH=${PATH}:`cat e_path.dat`
```

Стандартные каталоги для Windows-версий: \Program Files\MultiOberon, \Program Files (x86)\MultiOberon. Пути к исполняемым файлам устанавливаются из переменных окружения, либо записываются в файлы e_path.dat (r_path.dat для 64-бит) и считываются следующим образом.

```
setlocal
set /p PATH1=< "%~dp0%\..\e_path.dat"
set PATH=%PATH1%;%PATH%
```

2.3 УСЛОВИЯ РЕЖИМА ИЗ СРЕДЫ BLACKBOX

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

2.4 СТРУКТУРА КАТАЛОГОВ

Структура каталогов начальной точки базового X86 состоит из каталогов Mod для исходных текстов модулей, Code для кодовых файлов и Sym для символьных файлов. Структура каталогов МультиОберона описана в приложении 1. Такая структура отражает версию ПО с мульти-платформами и мульти-бэкендами.

Структура каталогов определяет размещение ПО, предназначенного для компиляции средствами МультиОберон.

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 КОМАНДЫ И ОПЦИИ КОМАНДНОЙ СТРОКИ

Команды и опции командной строки могут задаваться при вызове компилятора из консоли.

```
ddag@sdl:~/scadi0$ ombc co -h -path `pwd` -spath "${BBPATH}" DpxchPoll2 DpxchUDP
2Channel DpxchMUDP2Channel DpxchProform2 DpxchTzCaesar DpxchHndImit DpxchUctest
DpxchBssLib DpxchBss DpxchHndBss
```

Рисунок 1 – Компиляция из консоли

Аналогично, команды и опции командной строки могут задаваться при вызове компилятора из среды BlackBox.

```
OmbCompiler.CompileThis -h DpxchPoll2 DpxchUDP2Channel DpxchMUDP2Channel DpxchProform2
DpxchTzCaesar DpxchHndImit DpxchUctest DpxchBssLib DpxchBss DpxchHndBss DpxchNw DpxchHndNw
```

Рисунок 2 – Компиляция из среды BlackBox

Команды консоли представляют собой первый параметр вызова (co – в приведенном примере), а команды среды – вызываемую строку в виде <Имя Модуля>.<Имя команды>. В приведенном примере используется OmbCompiler.CompileThis.

Имя Модуля состоит из имени подсистемы и имени файла. Имя подсистемы начинается с первой заглавной буквы, имя файла начинается со второй заглавной буквы. Для модуля DpxchHndBss имя подсистемы будет Dpxch, имя файла будет HndBss.

Опции командной строки состоят из выражений типа –option для одинарных и –option value для двойных.

Приложение 2 содержит перечень опций командной строки. Приложение 3 содержит перечень команд компилятора.

Для компиляции из командной строки вводится имя компилятора (таблица 1-2), имя команды и список опций командной строки.

Для компиляции из среды BlackBox вводится символ коммандера !, <Имя Модуля>.<Имя команды> и список опций командной строки.

3.2 OMB – НАТИВНАЯ КОМПИЛЯЦИЯ В X86

3.2.1 Консольный режим

Простейший способ компиляции и запуска рассмотрим на примере модуля OmtestHelloWorld, приложение 4. Компиляция и запуск модуля на исполнение выполняется одной командой.

```
>ombc ex OmtestHelloWorld
Hello, World
```

Команда ex[ecute] по сути дела выполняет последовательно команды co[mpile] и r[un]. Обычно компиляция производится отдельно.

```
>ombc co OmtestHelloWorld
omb:compiling /home/ddag/scadi0/Omtest/Mod/HelloWorld.mob
>/usr/local/bin/multioberon/Omtest/Code/HelloWorld.ocf code=52 glob=0
```

Команда запуска динамически загружает и выполняет модуль OmtestHelloWorld.

```
>ombc run OmtestHelloWorld
```

```
Hello, World
```

Исходный код файла модуля хранится в каталоге Подсистема/Mod/ с именем Файл.mob. Для имени модуля OmtestHelloWorld исходный код хранится в Omtest/Mod/HelloWorld.mob. Файл с расширением mob является текстовым и доступен для редактирования любым инструментом. Файлы с расширением odc, отредактированные в среде BlackBox, являются бинарными. Для компиляции таких файлов используется опция –odc.

```
>ombc co –odc OmtestHelloWorld
```

Исходный код в таком случае берется из каталога Подсистема/Mod/ с именем Файл.odc. Для приведенного примера берется Omtest/Mod/HelloWorld.odc.

При компиляции создаются кодовый и символьный файлы. Кодовый файл записывается как Omtest/Sym/HelloWorld.ocf. Каталоги кодовых файлов для различных бэкендов приведены в таблице 1-3. Символьный файл записывается как Omtest/Sym/HelloWorld.osf. Каталоги символьных файлов для различных бэкендов приведены в таблице 1-4. Кодовый файл представляет собой бинарный файл в формате, близком к объектному. Символьный файл представляет собой бинарный файл в формате, содержащем список символьных имен.

Также МультиОберон добавляет файлы использования *.ouf, которые содержат списки импорта и информацию об ограничениях. Эти файлы расположены в Sym или S[backend][os][arch] каталогах.

Опция –pl 0 подавляет печать сообщения вида «omb:compiling …», оставляя при этом возможность вывода сообщений об ошибках.

Остальные опции командной строки не используются в компиляторе Omb.

Таким образом, компиляция в консольном режиме осуществляется командой.

```
ombc co ?-odc? ?-pl 0? module ?module? ?module?
```

Примером может служить следующая команда.

```
ombc co -pl 0 OmtestHelloWorld OmtestFormats
```

Запуск и запуск с предварительной компиляцией в консольном режиме осуществляется соответственно командами.

```
ombc run ?-odc? module
ombc ex ?-odc? module
```

3.2.2 Режим из среды BlackBox

Вызов из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❶ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

```
❶ OmbCompiler.CompileThis OmtestHelloWorld OmtestFormats OmtestDateTime OmtestMkTraps
OmtestHeap◢
```

Используемые опции аналогичны консольному режиму.

Таким образом, компиляция в среде BlackBox осуществляется командой.

```
^Q OmbCompiler.CompileThis ?-odc? ?-pl 0? module ?module? ?module?
```

Запуск откомпилированных модулей без параметров осуществляется указанием строки в виде <Имя Модуля>.<Имя команды>. Запуск откомпилированных модулей с параметрами осуществляется согласно соглашениям BlackBox с указанием строковых параметров в кавычках.

3.3 OMB – ЛИНКОВКА X86 ПРИЛОЖЕНИЙ

3.3.1 Консольный режим

Линковка используется для создания исполняемого файла для последующего запуска.

Простейший способ линковки исполняемого файла также рассмотрим на примере модуля OmtestHelloWorld, приложение 4. Создание исполняемого файла выполняется одной командой.

```
>ombc link -r OmtestHelloWorld
```

Для проверки наберите имя файла.

```
>OmtestHelloWorld
Hello, World
```

Исполняемый файла модуля хранится в каталоге Подсистема/Bin[os][arch]/ с именем Файл или Файл.exe для Windows. Для указания места создания исполняемого файла используется опция –o.

```
>ombc link -r –o “sbinbwe/OHelloWorld” OmtestHelloWorld
```

Опция –r использует рекурсивный автоматический импорт. Без ее использования потребуется явно указывать список импортируемых модулей в порядке импорта.

```
>ombc link -r –o “sbinbwe/OHelloWorld” Kernel Log Math Strings OStrings OLog
HostConLog Runner OmtestHelloWorld
```

3.3.2 Режим из среды BlackBox

Запуск линковщика из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❗ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

```
4. Link exe if you want to call them from console
❗ OmbLinker.LinkExe -tl 2 -r OmtestHelloWorld◢
```

Используемые опции аналогичны консольному режиму.

Таким образом, линковка в среде BlackBox осуществляется командой.

```
^Q OmbLinker.LinkExe ?-r? ?-o outfile? module ?module? ?module?
```

При указании опции рекурсивного импорта указывается только один головной модуль.

3.4 OMF – ТРАНСЛЯЦИЯ В ЯЗЫК С И СБОРКА ОБЪЕКТНЫХ ФАЙЛОВ

3.4.1 Консольный режим

Компиляция модуля OmtestHelloWorld бэкендом Omf приводит к генерации С-кода транслятором.

```
>omfc co OmtestHelloWorld
omf:compiling                          C:\DVDagaev\dsu/Omtest/Mod/HelloWorld.mob
>C:\DVDagaev\dsu/Omtest/Cfwe/OmtestHelloWorld .c=2184 .h=0
```

Исходный код файла модуля хранится в каталоге Подсистема/Mod/ с именем Файл.mob. Для имени модуля OmtestHelloWorld исходный код хранится в Omtest/Mod/HelloWorld.mob. Файл с расширением mob является текстовым и доступен для редактирования любым инструментом. Файлы с расширением odc, отредактированные в среде BlackBox, являются бинарными. Для компиляции таких файлов используется опция –odc.

```
>omfc co –odc OmtestHelloWorld
```

Сгенеренный С-код доступен в каталоге Подсистема/C[backend][os][arch]/ (таблица 1-3 приложения 1). Для 32-битной Windows имеем каталог Omtest/Cfwe. Сгенеренный код имеет имя, квалифицированное именем подсистемы, для нашего примера OmtestHelloWorld.c. Сгенеренный для примера HelloWorld из приложения 4 код представлен в приложении 5.

При компиляции также создается символьный файл. Эти файлы расположены в Sym=S[backend][os][arch] каталогах. Символьный файл записывается как Omtest/Sfwe/HelloWorld.osf. Каталоги символьных файлов для различных бэкендов приведены в таблице 1-4. Символьный файл представляет собой бинарный файл в формате, содержащем список символьных имен.

Для использования программы в качестве динамически загружаемого модуля необходимо после операции трансляции запустить команду build для сгенеренных файлов.

```
>omfc build -pl 2 OmtestHelloWorld
Omtest/Cfwe>clang   -m32   -O2   -c   OmtestHelloWorld.c   -I   ../../C   -I
../../System/Cfwe -I ../../Host/Cfwe -I ../../Omtest/Cfwe
```

Команда сборки build запускает внешний компилятор, указанный в опциях Omf.cfg. В качестве внешнего компилятора может использоваться gcc (если задано cc=gcc) или clang (если задано cc=clang). Опция –pl 2 устанавливает PRINT_LEVEL_NORM, при задании такого уровня консольная строка с параметрами компилятора выводится на экран перед выполнением.

При сборке создаются объектные (кодовые) файлы. Эти файлы расположены в Code=C[backend][os][arch] каталогах. Объектный файл записывается как Omtest/Cfwe/HelloWorld.o. Каталоги кодовых (объектных) файлов для различных бэкендов приведены в таблице 1-3.

Команда выполнения ex[ecute] по сути дела выполняет последовательно команды co[mpile], bu[ild] и r[un]. Обычно компиляция производится отдельно.

Также МультиОберон добавляет файлы использования *.ouf, которые содержат списки импорта и информацию об ограничениях. Эти файлы расположены в Sym или S[backend][os][arch] каталогах.

Опция –pl 0 подавляет печать сообщения вида «omf:compiling …», оставляя при этом возможность вывода сообщений об ошибках.

Опция -32 предназначена для сборки 32-битного кода 64-битным компилятором. По умолчанию создается код в размерности компилятора.

Опция -64 предназначена для сборки 64-битного кода 32-битным компилятором. По умолчанию создается код в размерности компилятора.

Остальные опции командной строки не используются в компиляторе Omf.

Таким образом, компиляция в консольном режиме осуществляется командой.

```
omfc co ?-odc? ?-pl 0? ?-32? ?-64? module ?module? ?module?
```

Примером может служить следующая команда.

```
omfc co -pl 0 OmtestHelloWorld OmtestFormats
```

Запуск объектного кода в консольном режиме осуществляется соответственно командой run.

```
>omfc run module
Hello, World
```

Выполнение модуля программы командой run приводит к загрузке и старту объектного файла модуля, в нашем случае Omtest/Cfwe/HelloWorld.o. После загрузки старт осуществляется с секции BEGIN.

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

Данной секции ставится в соответствие код C-программы export int main. Данный код при детализации добавляет секции импорта и инициализации импортируемых модулей.

Для 32-битной архитектуры ArmV71 используются каталоги Cfu4 для кодовых файлов, Sfu4 для символьных файлов и Binu4 для приложений. Для 64-битной архитектуры Aarch64 используются каталоги Cfu8 для кодовых файлов, Sfu8 для символьных файлов и Binu8 для приложений.

3.4.2 Режим из среды BlackBox

Вызов из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❗ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

```
❗ OmfCompiler.CompileThis -64 OmtestHelloWorld OmtestFormats OmtestDateTime OmtestMkTraps
OmtestHeap◢
```

Используемые опции аналогичны консольному режиму, в частности опция -64 указывает на необходимость сборки 64-битного приложения 32-битной средой BlackBox.

Таким образом, компиляция в среде BlackBox осуществляется командой.

```
^Q OmfCompiler.CompileThis ?-odc? ?-pl 0? ?-32? ?-64? module ?module? ?module?
```

Откомпилированные Omf-модули не запускаются из среды BlackBox. Для этого используется команда run консольного режима.

3.5 OMF – ЛИНКОВКА ПРИЛОЖЕНИЙ

3.5.1 Консольный режим

Линковка используется для создания исполняемого файла из собранных командой build объектных файлов для последующего запуска.

Простейший способ линковки исполняемого файла также рассмотрим на примере модуля OmtestHelloWorld, приложение 4. Создание исполняемого файла выполняется одной командой.

```
>omfc link -r OmtestHelloWorld
```

Для проверки наберите имя файла.

```
>OmtestHelloWorld
Hello, World
```

Исполняемый файла модуля хранится в каталоге Подсистема/Bin[os][arch]/ с именем Файл или Файл.exe для Windows. Для указания места создания исполняемого файла используется опция –o.

```
>omfc link -r –o “sbinfwe/OHelloWorld” OmtestHelloWorld
```

Опция –r использует рекурсивный автоматический импорт. Без ее использования потребуется явно указывать список импортируемых модулей в порядке импорта.

```
>omfc link -r –pl 2 –o “sbinfwe/OHelloWorld” Kernel Log Math Strings OStrings
OLog HostConLog Runner OmtestHelloWorld
gcc      OmtestHelloWorld.o      -o      ../../Omtest/Cfwe/OmtestHelloWorld.exe
../../Host/Cfwe/HostConLog.o -m32 -O2
```

Опция –pl 2 устанавливает PRINT_LEVEL_NORM, при задании такого уровня консольная строка с параметрами компилятора выводится на экран перед выполнением. Консольная строка линковки использует заданный в конфигурации Omf.cfg линкер и опции, как-то:

```
cc=gcc
lnk=gcc
gcc_opt=-O2
gcc_lnkopt=-O2
```

Опция -32 предназначена для линковки 32-битного кода 64-битным компилятором. По умолчанию линкуется код в размерности компилятора.

Опция -64 предназначена для линковки 64-битного кода 32-битным компилятором. По умолчанию линкуется код в размерности компилятора.

Для 32-битной архитектуры ArmV71 используются каталоги Cfu4 для кодовых файлов, Sfu4 для символьных файлов и Binu4 для приложений. Для 64-битной архитектуры Aarch64 используются каталоги Cfu8 для кодовых файлов, Sfu8 для символьных файлов и Binu8 для приложений.

3.5.2 Режим из среды BlackBox

Запуск линковщика из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❗ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

❗ OmfLinker.LinkExe -tl 2 -pl 2 -r OmtestHelloWorld◢

Используемые опции аналогичны консольному режиму.

Таким образом, линковка в среде BlackBox осуществляется командой.

```
^Q OmfLinker.LinkExe ?-r? ?-o outfile? ?-64? ?-pl 0? module ?module? ?module?
```

При указании опции рекурсивного импорта указывается только один головной модуль. Опция -64 может применяться для указания необходимости сборки 64-битного приложения 32-битной средой BlackBox.

3.6 OML – ГЕНЕРАЦИЯ КОДА LLVM И СБОРКА ОБЪЕКТНЫХ ФАЙЛОВ

3.6.1 Консольный режим

Компиляция модуля OmtestHelloWorld бэкендом Oml приводит к генерации LLVM-кода в текстовом *.ll или бинарном .bc виде.

```
>omlc co OmtestHelloWorld
```

```
oml:compiling ./Omtest/Mod/HelloWorld.mob
>./Omtest/Clue/OmtestHelloWorld .ll=10014 .bc=3544
```

Исходный код файла модуля хранится в каталоге Подсистема/Mod/ с именем Файл.mob. Для имени модуля OmtestHelloWorld исходный код хранится в Omtest/Mod/HelloWorld.mob. Файл с расширением mob является текстовым и доступен для редактирования любым инструментом. Файлы с расширением odc, отредактированные в среде BlackBox, являются бинарными. Для компиляции таких файлов используется опция –odc.

```
>omlc co –odc OmtestHelloWorld
```

Сгенеренный LLVM-код доступен в каталоге Подсистема/C[backend][os][arch]/ (таблица 1-3 приложения 1). Для 32-битной Unix системы имеем каталог Omtest/Clue. Сгенеренный код имеет имя, квалифицированное именем подсистемы, для нашего примера OmtestHelloWorld.ll, OmtestHelloWorld.bc. Текстовый код .ll используется разработчиком и мало информативен для пользователя.

При компиляции также создается символьный файл. Эти файлы расположены в Sym=S[backend][os][arch] каталогах. Символьный файл записывается как Omtest/Slue/HelloWorld.osf. Каталоги символьных файлов для различных бэкендов приведены в таблице 1-4. Символьный файл представляет собой бинарный файл в формате, содержащем список символьных имен.

Для использования программы в качестве динамически загружаемого модуля необходимо после операции трансляции запустить команду build для создания объектных файлов из сгенеренных.

```
>omlc build -pl 2 OmtestHelloWorld
      /usr/local/bin/multioberon/Binue/llc   -filetype=obj   -O0   -march=x86
OmtestHelloWorld.ll -o OmtestHelloWorld.o
```

Команда сборки build запускает внешнюю LLVM-утилиту llc для создания объектных файлов. Опция –pl 2 устанавливает PRINT_LEVEL_NORM, при задании такого уровня консольная строка с параметрами компилятора выводится на экран перед выполнением. Консольная строка линковки использует заданный в конфигурации Oml.cfg компилятор и опции, как-то:

```
llc=/usr/local/bin/multioberon/Binue/llc
llc_opt=-O0 -march=x86
```

При сборке создаются объектные (кодовые) файлы. Эти файлы расположены в Code=C[backend][os][arch] каталогах. Объектный файл записывается как Omtest/Clue/HelloWorld.o. Каталоги кодовых (объектных) файлов для различных бэкендов приведены в таблице 1-3.

Команда выполнения ex[ecute] по сути дела выполняет последовательно команды co[mpile], bu[ild] и r[un]. Обычно компиляция производится отдельно.

Также МультиОберон добавляет файлы использования *.ouf, которые содержат списки импорта и информацию об ограничениях. Эти файлы расположены в Sym или S[backend][os][arch] каталогах.

Опция –pl 0 подавляет печать сообщения вида «oml:compiling …», оставляя при этом возможность вывода сообщений об ошибках.

Опция -32 предназначена для сборки 32-битного кода 64-битным компилятором. По умолчанию создается код в размерности компилятора.

Опция -64 предназначена для сборки 64-битного кода 32-битным компилятором. По умолчанию создается код в размерности компилятора.

Остальные опции командной строки не используются в компиляторе Oml.

Таким образом, компиляция в консольном режиме осуществляется командой.

```
omlc co ?-odc? ?-pl 0? ?-32? ?-64? module ?module? ?module?
```

Примером может служить следующая команда.

```
omlc co -pl 0 OmtestHelloWorld OmtestFormats
```

Запуск объектного кода в консольном режиме осуществляется соответственно командой run.

```
>omlc run module
Hello, World
```

Выполнение модуля программы командой run приводит к загрузке и старту объектного файла модуля, в нашем случае Omtest/Clue/HelloWorld.o. После загрузки старт осуществляется с секции BEGIN.

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

Для 32-битной архитектуры ArmV71 используются каталоги Clu4 для кодовых файлов, Slu4 для символьных файлов и Binu4 для приложений. Для 64-битной архитектуры Aarch64 используются каталоги Clu8 для кодовых файлов, Slu8 для символьных файлов и Binu8 для приложений.

3.6.2 Режим из среды BlackBox

Вызов из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❗ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

```
❗ OmlCompiler.CompileThis -64 -options lbv OmtestHelloWorld OmtestFormats OmtestDateTime
OmtestMkTraps OmtestHeap◢
```

Используемые опции аналогичны консольному режиму, в частности опция -64 указывает на необходимость сборки 64-битного приложения 32-битной средой BlackBox. Опции –options lbv указывают на необходимость генерации ll-файла (опция l), bc-файла (опция b) и верификации (опция v). По умолчанию задается значение –options lb.

Таким образом, компиляция в среде BlackBox осуществляется командой.

```
^Q OmlCompiler.CompileThis ?-odc? ?-pl 0? ?-32? ?-64? ?-options? module ?module?
?module?
```

Откомпилированные Oml-модули не запускаются из среды BlackBox. Для этого используется команда run консольного режима.

3.7 OML – ЛИНКОВКА ПРИЛОЖЕНИЙ

3.7.1 Консольный режим

Линковка используется для создания исполняемого файла из собранных командой build объектных файлов для последующего запуска.

Простейший способ линковки исполняемого файла также рассмотрим на примере модуля OmtestHelloWorld, приложение 4. Создание исполняемого файла выполняется одной командой.

```
>omlc link -r OmtestHelloWorld
Omtest/Clue>gcc   OmtestHelloWorld.o   -o   ../../Omtest/Clue/OmtestHelloWorld
../../System/Clue/OStrings.o                          ../../System/Clue/OLog.o
../../Host/Clue/HostConLog.o                        ../../System/Clue/Kernel.o
../../System/Clue/Runner.o -m32 -O0 -lm -ldl
```

Для проверки наберите имя файла.

```
>OmtestHelloWorld
Hello, World
```

Исполняемый файла модуля хранится в каталоге Подсистема/Bin[os][arch]/ с именем Файл или Файл.exe для Windows. Для указания места создания исполняемого файла используется опция –o.

```
>omlc link -r –o “sbinlue/OHelloWorld” OmtestHelloWorld
```

Опция –r использует рекурсивный автоматический импорт. Без ее использования потребуется явно указывать список импортируемых модулей в порядке импорта.

Опция –pl 2 устанавливает PRINT_LEVEL_NORM, при задании такого уровня консольная строка с параметрами компилятора выводится на экран перед выполнением. Консольная строка линковки использует заданный в конфигурации Oml.cfg линкер и опции, как-то:

```
lnk=gcc
gcc_lnkopt=-O0 -lm -ldl
```

В качестве линкера используются средства стандартных gcc и clang. Опция -32 предназначена для линковки 32-битного кода 64-битным компилятором. По умолчанию линкуется код в размерности компилятора.

Опция -64 предназначена для линковки 64-битного кода 32-битным компилятором. По умолчанию линкуется код в размерности компилятора.

Для 32-битной архитектуры ArmV71 используются каталоги Clu4 для кодовых файлов, Slu4 для символьных файлов и Binu4 для приложений. Для 64-битной архитектуры Aarch64 используются каталоги Clu8 для кодовых файлов, Slu8 для символьных файлов и Binu8 для приложений.

3.7.2 Режим из среды BlackBox

Запуск линковщика из среды BlackBox осуществляется посредством строки, начинающейся символом коммандера ❗ (задаваемого ^Q) и заканчивающимся символом окончания строки ◢.

Примером может служить следующая команда.

❗ OmlLinker.LinkExe -tl 2 -r OmtestHelloWorld◢

Используемые опции аналогичны консольному режиму.

Таким образом, линковка в среде BlackBox осуществляется командой.

```
^Q OmlLinker.LinkExe ?-r? ?-o outfile? ?-64? ?-pl 0? module ?module? ?module?
```

При указании опции рекурсивного импорта указывается только один головной модуль. Опция -64 может применяться для указания необходимости сборки 64-битного приложения 32-битной средой BlackBox.

3.8 ФАЙЛЫ ИСПОЛЬЗОВАНИЯ И ДЕРЕВО ИМПОРТА

МультиОберон добавляет файлы использования *.ouf, которые содержат списки импорта и информацию об ограничениях. Эти файлы расположены в Sym или S[backend][os][arch] каталогах. Бинарный файл использования имеет следующий формат (item, value)* :

- item – целочисленный номер опции;
- value – строковая константа как значение опции.

Опции, используемые в текущей версии, приведены в таблице 1:

Номер	Значение	Описание
16	sNAME	Имя модуля

50	sIMPNAME	Имя импортируемого модуля
51	sIMALIAS	Псевдоним импортируемого модуля
52	sLIBCODE	Имя библиотеки в модуле с кодом
53	sLIBNOCODE	Имя библиотеки в модуле без кода

Таблица 1 – Опции файлов использования .ouf

Файлы использования позволяют обходить (операция - траверс) все дерево импортируемых модулей. Пример – команда tr[averse]:

```
>Binwe\ombc trav OmtestHelloWorld
>Binue/ombc trav OmtestHelloWorld
OmtestHelloWorld
  Runner
    SYSTEM
    ?c:\suok5\dsu/System/Slwe/SYSTEM.ouf
    Kernel
      Api
      -Api [libcmt]
      OLog
        OStrings
        -OStrings
      -OLog
    -Kernel
  -Runner
-OmtestHelloWorld
```

Модуль Runner импортируется из OmtestHelloWorld, модули SYSTEM и Kernel импортируются из Runner, модули Api и OLog импортируются из Kernel. Модуль SYSTEM никогда не компилировался, поэтому файл SYSTEM.ouf отсутствует. Модуль Api имеет библиотеку ‘libcmt’.

Следует заметить, что импортная информация специфична для бэкенда, она может располагаться в файлах .ocf для BlackBox и .c для Ofront. Для избежания специфического, требующего времени, парсинга при получении списков импорта модулей было решено добавить явно файлы .ouf.

Алгориты обхода дерева импорта широко используются в рекурсивных сборках и линковках при включении опции –r.

3.9 ВОЗМОЖНОСТИ ТЕСТИРОВАНИЯ

Тестовые возможности МультиОберона покрывают как отдельные модули, так и компиляторв целом. Например, тестируется модуль OmtestSimple с процедурой PROCEDURE Sum(x, y: INTEGER): INTEGER. Нам потребуется модуль [module_name]_test – OmtestSimple_test.

```
MODULE OmtestSimple_test;
IMPORT T := Testing, OmtestSimple;
PROCEDURE Test0Basic* (VAR rec: T.Rec);
      BEGIN
            CASE rec.n_test OF
            | 0:
                  rec.res_type := T.RES_INT;
                  rec.msg := 'Sum x+x';
                  rec.i_req := 6;
                  rec.i_res := OmtestSimple.Sum(3, 0);
            | 1:
                  rec.res_type := T.RES_INT;
```

```
                rec.msg := 'Sum x+y';
                rec.i_req := 5;
                rec.i_res := OmtestSimple.Sum(3, 2);
            | 2:
                rec.finish := TRUE;
            ELSE
            END;
      END Test0Basic;
END OmtestSimple_test.
```

Модуль OmtestSimple_test реализует набор процедур с шаблоном Test*, вызываемых тестовым окружением. Соглашение по именам Test0*, Test1* используется для вывода результатов тестирования в отсортированном по возрастанию порядке.

Каждый тест устанавливает поля структуры Testing.Rec. Тип возвращаемого значения res_type устанавливается первым, чтобы показать, какие поля используются. Сообщение пользователю msg выводится при тестировании. Для каждого типа результат сравнивается с требуемой величиной. В нашем случае i_res представляет собой целочисленный результат, i_req является целочисленной требуемой величиной.

Тестовое окружение обнуляет структуры перед каждым вызовом, только номер сета и номер теста (n_test) устанавливаются предварительно. В случае несоответствия сообщение об ошибке быводится тестовой системой.

Тесты компилятора могут загружаться и выполняться динамически. Так что модули должны быть предварительно подготовлены для динамической загрузки:

```
>Binwe\ombc co -odc OmtestSimple OmtestSimple_test
>Binue/ombc co -odc OmtestSimple OmtestSimple_test
```

После компиляции используйте команду test в консольном компиляторе:

```
>Binwe\ombc test OmtestSimple
>Binue/ombc test OmtestSimple
[ALL] ========== Total 2 tests, 0 bad, result= 100.0%
```

Опция уровня печати –pl дает больше информации:

```
>Binwe\ombc test –pl 3 OmtestSimple
>Binue/ombc test –pl 3 OmtestSimple
[OmtestSimple_test.Test0Basic.000] INT i_res= 6 i_req= 6 :Sum x+x
[OmtestSimple_test.Test0Basic.001] INT i_res= 5 i_req= 5 :Sum x+y
[OmtestSimple_test.] ---------- In module 1 sets, 2 tests, 0 bad
[ALL] ========== Total 2 tests, 0 bad, result= 100.0%
```

Специальные тесты для компилятора также прилагаются:

```
B\bwe_tests_tomake.bat       (* X86 *)
B\bwr_tests_tomake.bat       (* X64 *)
B/bue_tests_tomake.sh             (* X86 *)
B/bur_tests_tomake.sh             (* X64 *)
B/bu4_tests_tomake.sh             (* ArmV71 *)
B/bu8_tests_tomake.sh             (* Aarch64 *)
```

Компилируемые тесты ниже (OmtestOmc*) не относятся к специфическим модулям, а к компилятору в целом:

```
Binwe\ombc     co     -odc     OmtestOmcSimple_test     OmtestOmcStrings_test
OmtestOmcSystem_test      OmtestOmcImports_test      OmtestOmcExtensions_test
OmtestOmcBound_test OmtestOmcAdvanced_test
```

Специальные тесты компилятора также имеют скрипт сборки. Выполнение их приводит к печати, аналогичной следующей:

```
bwe_tests_run.bat
```

```
bue_tests_run.sh
c:\suok5\dsu>Binwe\ombc test -pl 2 OmtestOmcSimple_test OmtestOmcStrings_test
OmtestOmcSystem_test      OmtestOmcImports_test      OmtestOmcExtensions_test
OmtestOmcBound_test OmtestOmcAdvanced_test
[OmtestOmcSimple_test.] ---------- In module 9 sets, 104 tests, 0 bad
[OmtestOmcStrings_test.] ---------- In module 5 sets, 64 tests, 0 bad
[OmtestOmcSystem_test.Test1Addr.028]  ?LONG  li_res=  12  li_req=  16  :Proper
position of ptr after long
[OmtestOmcSystem_test.Test1Addr.029] ?LONG li_res= 16 li_req= 24 :SIZE of Rec
with LONGINT aligned by 8
[OmtestOmcSystem_test.Test1Addr.030]  ?LONG  li_res=  12  li_req=  16  :Proper
position of ptr after Rec with long
[OmtestOmcSystem_test.Test1Addr.031] ?LONG li_res= 12 li_req= 16 :SIZE of Rec
with LONGINT and char
[OmtestOmcSystem_test.Test1Addr.032] ?LONG li_res= 16 li_req= 24 :SIZE of Rec
with Rec with LONGINT aligned by 8
[OmtestOmcSystem_test.] ---------- In module 3 sets, 49 tests, 5 bad
[OmtestOmcImports_test.] ---------- In module 6 sets, 59 tests, 0 bad
[OmtestOmcExtensions_test.] ---------- In module 10 sets, 80 tests, 0 bad
[OmtestOmcBound_test.] ---------- In module 6 sets, 95 tests, 0 bad
[OmtestOmcAdvanced_test.] ---------- In module 6 sets, 42 tests, 0 bad
[ALL] ========== Total 493 tests, 5 bad, result= 98.98580121703854%
```

Тесты для JIT-компилятора для Oml LLVM могут запускаться скриптами:

```
lwe_tests_jit_run.bat
lue_tests_jit_run.sh
```

Приводимые тесты компилятора также могут быть статически собраны в приложение OmtestOmcCompiler и запущены:

```
Omtest\Cfwe\OmtestOmcCompiler –pl 2
Omtest/Cfue/OmtestOmcCompiler –pl 2
```

Список непрошедших тестов демонстрирует качество специфического бэкенда. Это используется для дальнейшего усовершенствования компилятора.

3.10 ВОЗМОЖНОСТИ БЕНЧМАРКИНГА

Данные функции используются для измерений времени работы выбранных процедур. Требуется модуль [module_name]_bench – например, OmtestBenchRoutines_bench для оценки OmtestBenchRoutines.

```
MODULE OmtestBenchRoutines_bench;
IMPORT T := Testing, OmtestBenchRoutines;
PROCEDURE BenchmarkPalindrome* (IN bench: T.Bench; VAR num_done: INTEGER);
      VAR count: INTEGER;
BEGIN
      num_done := 0; count := 0;
      WHILE num_done < bench.num DO
            IF OmtestBenchRoutines.IsPalindrome(
"A man, a plan, a canal: Panama") THEN
                  INC(count)
            END;
            INC(num_done)
      END
END BenchmarkPalindrome;
END OmtestBenchRoutines_bench.
```

Модуль OmtestBenchRoutines_bench реализует набор процедур Benchmark*, вызываемых тестовым окружением. Каждая такая процедура на вход получает Testing.Bench. Параметр bench.num устанавливает требуемое число итераций. Выходной параметр num_done устанавливает реальное число итераций для контроля во избежании оптимизации реального числа итераций выполнения.

Временные тесты могут загружаться и выполняться динамически. Модули должны быть предварительно подготовлены для динамической загрузки:

```
>Binwe\ombc co -odc OmtestBenchRoutines OmtestBenchRoutines_bench
>Binue/ombc co -odc OmtestBenchRoutines OmtestBenchRoutines_bench
```

После компиляции используется команда bench в консоли компилятора:

```
>Binwe\ombc bench OmtestBenchRoutines
>Binue/ombc bench OmtestBenchRoutines
[OmtestBenchRoutines_bench.BenchmarkPalindrome] 1000000  00:00:00.282000 282.0
ns/op
```

Тестовое окружение информирует, что весь 1000000 итераций был выполнен за 282 миллисекунды. Среднее время составило 282.0 наносекунд на операцию.

4 СООБЩЕНИЯ ОПЕРАТОРУ

4.1 ЗАДАНИЕ УРОВНЯ ПЕЧАТИ

Задание уровня печати осуществляется опцией –pl. Возможные значения уровня печати приведены в таблице 2-3. Значение по умолчанию равно 1.

Уровень печати необходим при выполнении внешних команд. При заданном значении >= 2 команда перед выполнением печатается на консоль. Пример приведен ниже.

```
omlc build -pl 2 OmtestHelloWorld
Host/Clue>/usr/local/bin/multioberon/Binue/llc -filetype=obj -O0 -march=x86
HostConLog.ll -o HostConLog.o
===== obj-building HostConLog ... done
Omtest/Clue>/usr/local/bin/multioberon/Binue/llc -filetype=obj -O0 -march=x86
OmtestHelloWorld.ll -o OmtestHelloWorld.o
===== obj-building OmtestHelloWorld ... done
```

4.2 ЗАДАНИЕ УРОВНЯ ТРАССИРОВКИ

Задание уровня трассировки осуществляется опцией –tl. Возможные значения уровня трассировки приведены в таблице 2-2. Значение по умолчанию равно 0.

Уровень трассировки необходим при необходимости выводить расширенную диагностическую печать компилятора на консоль.

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 – ОПЦИИ КОМАНДНОЙ СТРОКИ

МультиОберон использует следующие опции командной строки, таблица 2-1.

Опция	Описание	Возможные значения
-odc	Выбор файла .odc вместо .mob	-
-32	32-битный режим	-
-64	64-битный режим	-
-os	Выбор ОС для кросс-компиляции	Windows|Linux
-r	Использование рекурсивного автоматического импорта	-
-h	Не включать HostConLog автоматически	-
-n	Перекомпилировать только новые файлы	-
-tl	Уровень трассировки	Таблица 2-2
-pl	Уровень печати	Таблица 2-3
-ht	Тип обработчика ошибок	Таблица 2-4
-wsd	Разрешение писать в каталог System	-
-o	Имя выходного файла	Имя файла
-path	Путь к импортируемым подсистемам	Каталоги для всех приложений
-spath	Теневой путь к импортируемым подсистемам	Каталоги для всех приложений

Таблица 2-1 – Перечень опций командной строки

Возможные значения уровня трассировки приведены в таблице 2-2. Значение по умолчанию равно 0.

Значение	Имя	Описание
0*	TRACE_LEVEL_NONE	Нет трассировки
1	TRACE_LEVEL_MIN	Печать сообщений об ошибках и важных сообщений
2	TRACE_LEVEL_NORM	Печать основных сообщений и дампов
3	TRACE_LEVEL_DETAILED	Детальная трассировка

Таблица 2-2 – Значения уровней трассировки

Возможные значения уровня печати приведены в таблице 2-3. Значение по умолчанию равно 1.

Значение	Имя	Описание
0	PRINT_LEVEL_NONE	Печать только сообщений об ошибках
1*	PRINT_LEVEL_DEFAULT	Печать выполняемой операции и сообщений об ошибках
2	PRINT_LEVEL_NORM	Подробная печать операции и сообщений об оибках
3	PRINT_LEVEL_DETAILED	Детальная печать

Таблица 2-3 – Значения уровней печати

Возможные значения типа обработчика ошибок приведены в таблице 2-4. Значение по умолчанию равно 0.

Значение	Имя	Описание
0*	HANDLER_TYPE_AUTO	Выбор обработчика по умолчанию
1	HANDLER_TYPE_DLINK	Обработчик с использованием структур DLink
2	HANDLER_TYPE_FP	Обработчик с использованием указателя фрейма FP
3	HANDLER_TYPE_STACK	Обработчик с алгоритмом анализатора стека

Таблица 2-4 – Значения типа обработчика ошибок

ПРИЛОЖЕНИЕ 3 – ПЕРЕЧЕНЬ КОМАНД КОМПИЛЯТОРА

Перечень команд компилятора МультиОберон представлен в таблице 3-1.

Из консоли	Из BlackBox	Описание
co[mpile]	Om?Compiler.CompileThis	Компиляция списка программных модулей
r[un]	--	Запуск на выполнение
ex[ecute]	--	Выполнение после компиляции и линковки
li[nk]	Om?Linker.LinkExe	Линковка исполняемого файла
tr[averse]	Om?Linker.TraverseModule	Рекурсивная печать списков импорта модуля
cl[ear]	Om?Linker.ClearFiles	Удаление сгенеренных файлов
bu[ild]	Om?Linker.BuildFiles	Сборка промежуточных файлов (кроме Omb)
te[st]	Om?Linker.TestModules	Запуск тестов модулей
be[nch]	Om?Linker.BenchModules	Прогон временных характеристик

Таблица 3-1 – Команды компилятора МультиОберон

ПРИЛОЖЕНИЕ 4 – 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 5 – СГЕНЕРЕННЫЙ С-КОД ДЛЯ HELLO WORLD

Ниже приведен сгенеренный код для OmtestHelloWorld. Помимо собственно кода в текте 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"
};
```

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
	изменен-ных	заменен-ных	новых	аннули-рованных				