Компилятор МультиОберон

Дагаев Дмитрий Викторович,
Главный Эксперт,
АО «Русатом Автоматизированные системы управления»
Консультант проекта Информатика-21

Концепция Хоара

«Там, где необходимо, должны быть разработаны расширения для конкретных аппаратных устройств и для конкретных приложений. Великая сила Паскаля в том и состоит, что в нем очень мало ненужных свойств и почти нет нужды в подмножествах. Вот почему этот язык достаточно силен, чтобы выдержать специализированные расширения — Concurrent Pascal для работы в реальном времени, Pascal+ для моделирования дискретных событий и UCSD-Pascal для микропроцессорных рабочих станций. Если бы только мы могли извлекать правильные уроки из прошлых успехов, нам не было бы нужды учиться на наших неудачах.»

Э.Хоар, лекция лауреатов премии Тьюринга.

Чемоданчик (по Кауфману) — многофункциональный инструмент

Чемоданчик становится полезным, когда комплектуется набором съемных насадок.

В структуре многофункциональной системы необходимо предусмотреть интерфейсы и механизмы расширения.

Мульти компилятор — многоцелевая система

МультиОберон это компилятор языка Оберон с различными сменными бэкендами:

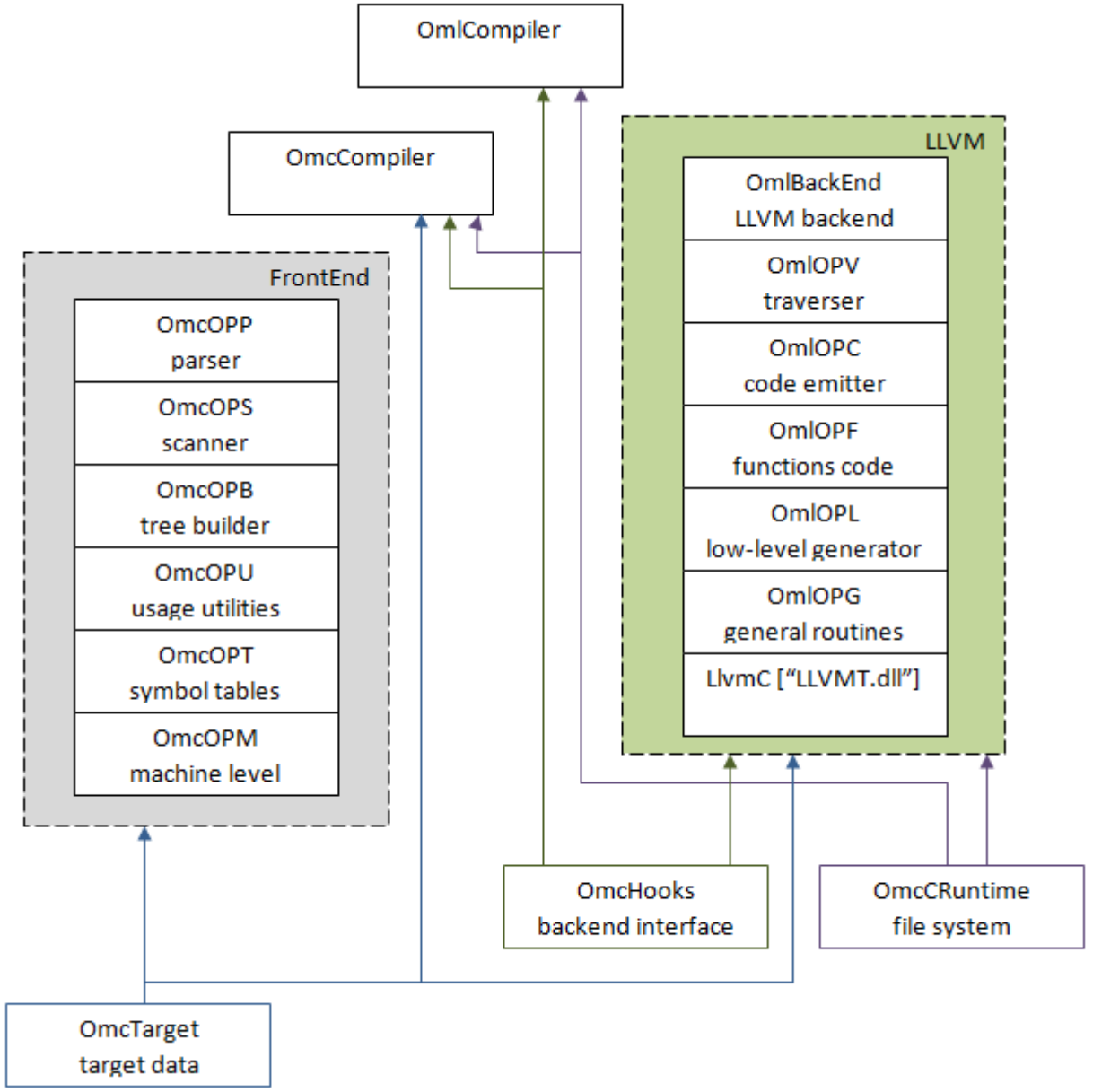
- Генератором нативного кода x86 для системы BlackBox;

- Транслятором Ofront в язык C;

- Генератором кода LLVM.

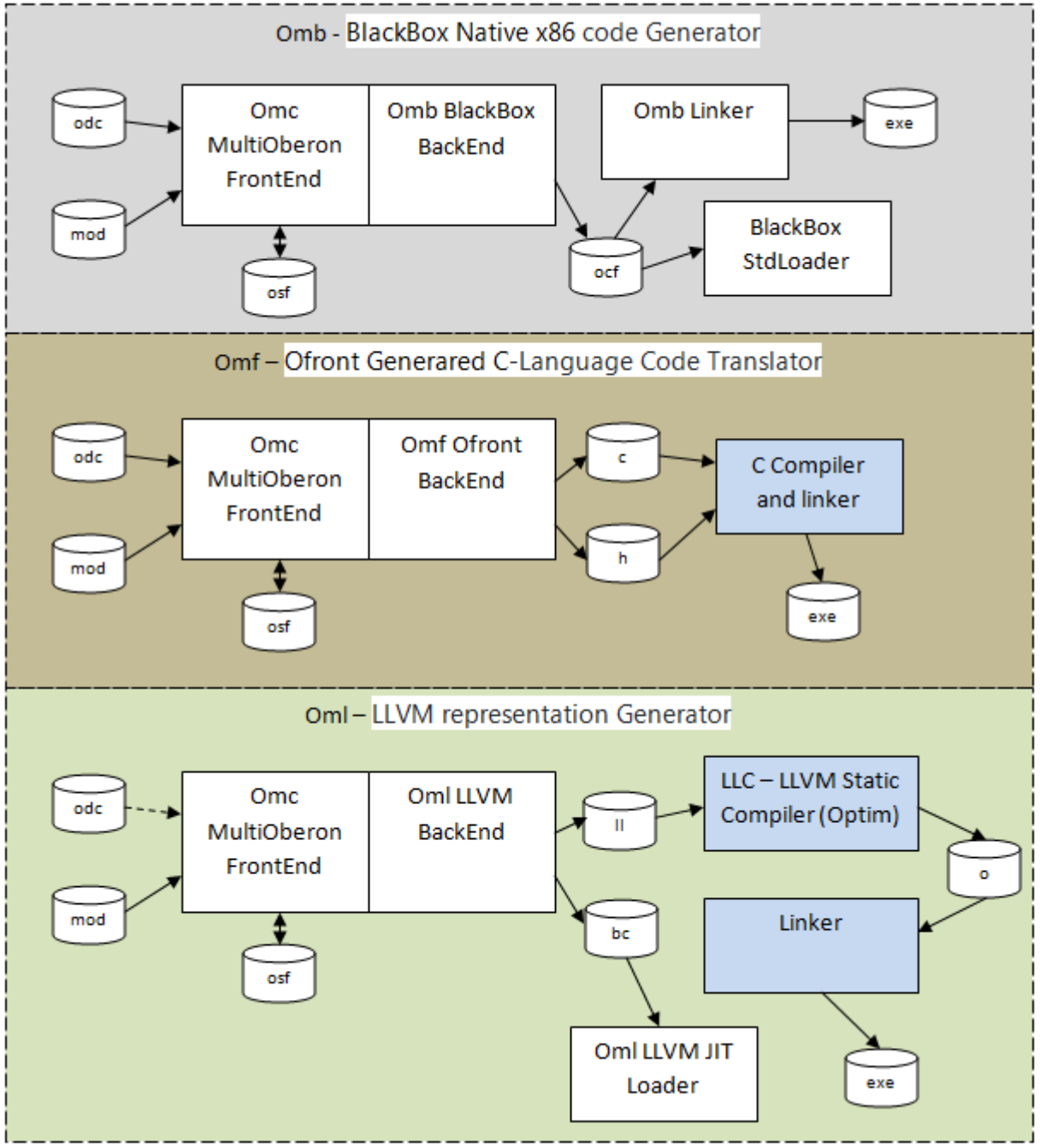
	OMB - BlackBox	OMF - Ofront	OML - LLVM
32 бит	YES	YES	YES
64 бит	NO	YES	YES
ARM	NO	YES	YES
Загрузка модулей	YES	NO	YES
Оптимизация	NO	YES	YES
Специальные ОС	NO	YES	NO

Oml - MultiOberon с бэкендом LLVM



- Модульная структура компилятора на основе OP2;
- Сменные бэкенды представляют собой сменные наборы модулей;
- LLVM бэкенд использует также библиотеку LLVMT.dll;
- Интерфейсы связи Hooks.BackEnd;
- Модуль целевых настроек Target.

Входные и выходные файлы



Omb генерит динамически загружаемый ocf;

Omf генерит .c, .h файлы, которые компилируются и линкуются сторонней программой;

Oml генерит llvm текстовый код *.ll и полученный из него бинарный бит-код *.bc;

Бинарный код может выполняться после загрузки OmlLoader, с LLVM JIT;

LLC статически компилирует в бинарий.

Варианты кодогенерации LLVM

- Использование промежуточного представления LLVM IR (intermediate representation) — нет гарантий совместимости;
- Использование бинарного представления — биткода LLVM bitcode — еще меньше гарантий совместимости;
- Использование библиотеки LLVM-C (C interface to LLVM http://llvm.org/doxygen/group__LLVMC.html) - выбрано.

LLVM.SetValueName – вызов функций.

```
MODULE LlvmC ["LLVMT.dll"];

TYPE

(* typedef struct LLVMOpaqueValue *LLVMValueRef; *)

ValueRef* = POINTER [untagged] TO RECORD END;

(* void LLVMSetValueName(LLVMValueRef Val, const char *Name); *)

PROCEDURE [ccall] SetValueName* ["LLVMSetValueName"] (val: ValueRef; name:
    CString);
```

Текущее состояние с оптимизацией

LLVM бэкенд в настоящее время генерит код без оптимизации. При необходимости его можно оптимизировать прилагаемой утилитой llc https://llvm.org/docs/CommandGuide/llc.html. Опции оптимизации llc -O=1 или 2, 3.

Работоспособность модулей с оптимизацией не проверялась, тесты для этого не писались.

Ofront — ясно, что C-код оптимизируется.

TODO — полная реализация оптимизации.

Мульти платформенность 32/64

Генерация кода для 32 и 64 бит, затрагивающая размерность указателей, RTTI, специальные функции;

Использование типа ADDRESS в SYSTEM.h для Ofront;

Существенно отличаются сборщики мусора в файлах Kernel для 64 и 32 битных версий;

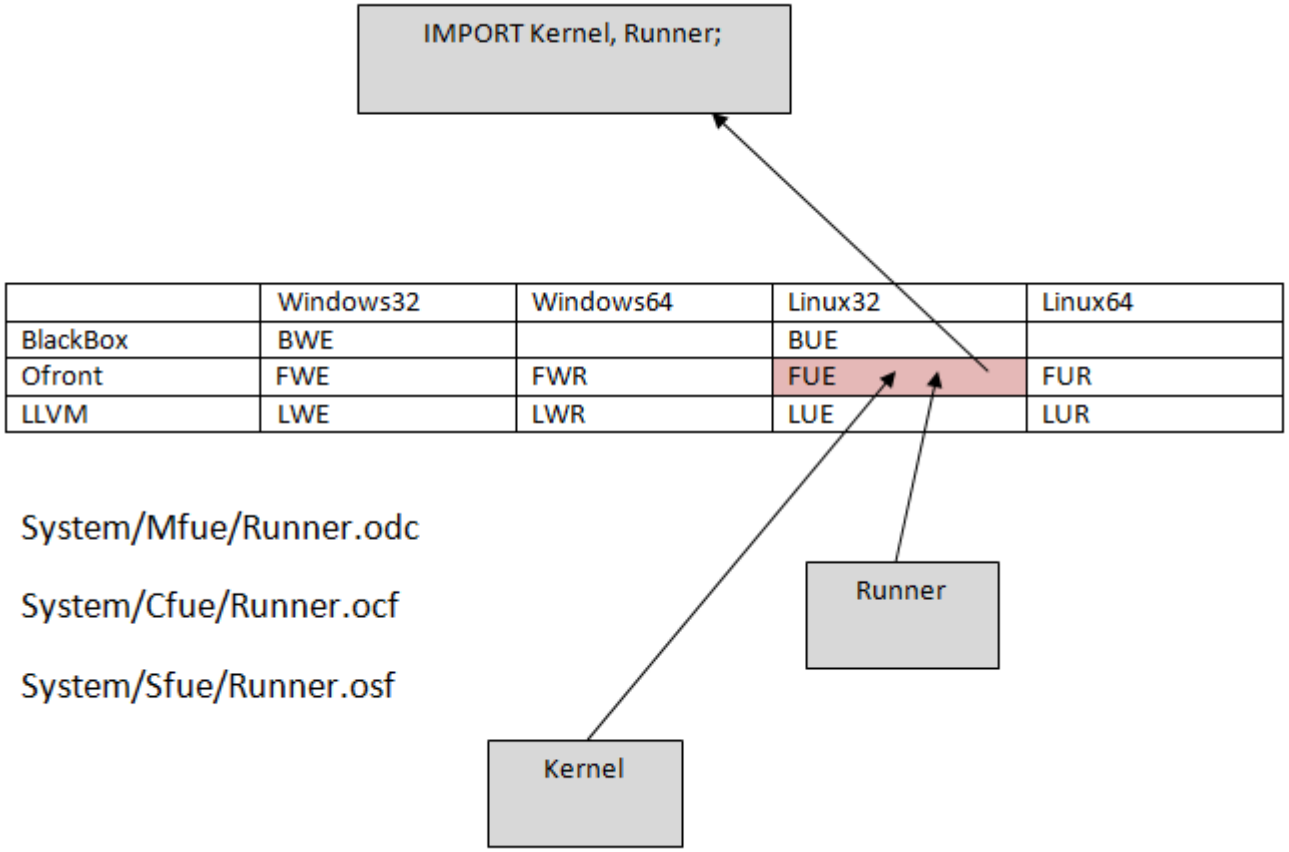
Заявлена (не опробована в Oml) поддержка LLVM для: Aarch64, AMDGPU, ARM, BPF, Hexagon, Lanai, MSP430, Mips, NVPTX, PowerPC, Sparc, SystemZ, X86, XCore.

Мульти платформенность в части ОС

В версии 0.9 Линкер переведен на реализацию Trurl для PE и ELF форматов;

Поддержка реализации разных оболочек для разных ОС.

Реализации Kernel, Runner и подсистема Host для всех ОС и бэкендов.



Мульти режим

Компилятор работает в двух режимах:

- Из среды BlackBox;
- Из командной строки — оболочками ombsh, omfsh, omlsh.

```
ombsh run OmtestHelloWorld
```

Входные файлы формата:

- odc — бинарный;
- mod — текстовый.

```
ombsh co -odc OmtestHelloWorld
```

Оболочки ombsh, omlsh реализуют динамическую загрузку модулей, не нужна среда разработки!

Динамическая загрузка и выполнение для LLVM

Выполнение из командной строки с помощью omlsh.

```
omlsh run OmtestHelloWorld
```

Этапы выполнения для каждого динамически подгружаемого `OmlLoader` модуля:

- Загрузка в память данных биткода *.bc;
- Парсинг биткода;
- Связывание переменных между модулями;
- Выполнение модуля с помощью компилятора MCJIT.

По предварительным оценкам, время загрузки составило 1330% от времени загрузки в BlackBox.

Поддержка различных модулей ядра BlackBox 1.6-1.7

Модуль Runner отделяет уровень представления работы (управления, загрузки) МультиОберона от платформо-специфических деталей (ОС, версия ядра, тип бэкенда).

Изменения центра в DevCPT — необходимость совместимости (sysflag) по Target.kernelVersion.

Для версии 1.6

```
Module* = POINTER TO RECORD
   [untagged]

    name-: Name;

END;

PROCEDURE (h: LoaderHook) ThisMod*
   (IN name: ARRAY OF SHORTCHAR):
   Module, NEW, ABSTRACT;
```

Для версии 1.7

```
Module* = POINTER TO RECORD
   [untagged]

    name-: Utf8Name;

END;

PROCEDURE (h: LoaderHook) ThisMod*
   (IN name: ARRAY OF CHAR): Module,
   NEW, ABSTRACT;
```

Абстрагирование от наследования в Log, Files?

- OLog отказ от зависимости наследования от Log.Hook для BlackBox и необходимости реализации специфических функций, например:

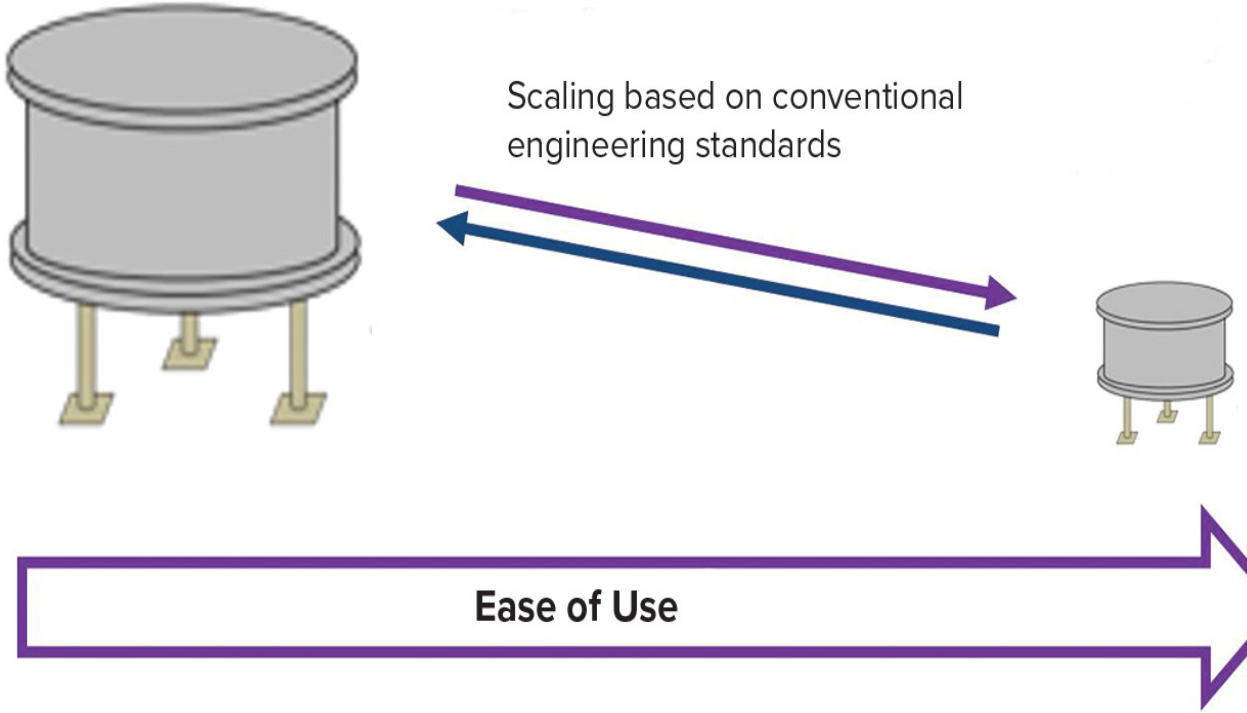
```
Log.ParamMsg(s, p0, p1, p2: ARRAY OF CHAR);

Log.ViewForm(v: ANYPTR; w, h: INTEGER);
```

- OLog – инкапсуляция Log и реализация дополнительных функций, например:

```
OLog.Adr(x: SYSTEM.PTR);

OLog.SString(str: ARRAY OF SHORTCHAR);
```

- Планируется переход от Files->OFiles, с учетом возможных реализаций Files с 64-битовыми длинами.

Мульти языковость



Масштабируемая технология с использованием Scale Down Model.

Демасштабирование излишней функциональности

Система ограничений с начальной точкой в виде синтаксиса Компонентного Паскаля:

Уровень –С- (Компонентный паскаль) -> 0 ограничений + Kernel уровня –С-;

Уровень -07- для последующей реализации;

Уровень -880- для последующей реализации.

Подстановка компиляторов для одной кодовой базы

Реализация с гарантией Liskov Substitution Principle. Замещение компиляторов по свойству допустимости входного кода:

*Если (компилятор) **S** является подтипом (компилятора) типа **T** , то возможно замещение компилятора **T** на компилятор **S.***

Отношение подтипов при ***T*** → ***S*** для одной и той же базы программ можно установить в том случае, когда имеется один frontend с переменной системой ***ограничений***. Множество свойств подтипа ***S*** должно включать множество свойств базового типа ***T,*** у которого ограничений больше.

Гарантия LSP — одно и то же абстрактное синтаксическое дерево, подаваемое на вход разным компиляторам.

Семантическое подобие O7 и существенные критерии

Оберон O7

```
BaseType* = RECORD END;

MyType* = RECORD (BaseType)
    num-: INTEGER;
    code-: LONGINT;
END;
```

Диагностирует синтаксическую ошибку на LONGINT.

Расширяемость по умолчанию более компактна.

Не совместима с синтаксисом Компонентного Паскаля.

МультиОберон

```
RESTRICT -LONGINT -SHORTINT;

BaseType* = EXTENSIBLE RECORD END;

MyType* = RECORD (BaseType)
    num-: INTEGER;
    code-: LONGINT;
END;
```

Главное - диагностирует ошибку наличия LONGINT.

Расширяемость записей с помощью EXTENSIBLE гарантирует LSP — семантически подобная O7 конструкция второстепенна.

Дейкстра об ограничениях

Говоря об умеренности, я имею в виду, к примеру, что не только выражение «for» из ALGOL 60, но и цикл «DO» из FORTRAN'а могут оказаться выкинутыми за свою причудливость.

RTTI и структуры данных Kernel уровня C

Генерация структур данных, соответствующих уровню –C-.

Вариативность структур данных в зависимости от разрядности и версии ядра.

Независимость структур данных от типа бэкенда (BlackBox, Ofront, LLVM).

Комплектация сгенеренными версиями ядра для различных платформ и модификаций.

Целесообразность использования ADRINT.

```
Module* = POINTER TO RECORD [untagged]
    next-: Module;
    opts-: SET;    (* 0..15: compiler opts, 16..31: kernel flags *)
    refcnt-: INTEGER;    (* <0: module invalidated *)
    compTime-, loadTime-: ARRAY 6 OF SHORTINT;
    body-: Command;    (* Module__body *)
    term-: Command;    (* Module__close *)
    nofimps-, nofptrs-: INTEGER;
    csize-, dsize-, rsize-: @ADDR;
    code-, data-, refs-: @ADDR;
    procBase-, varBase-: @ADDR;    (* meta base addresses *)
    names-: POINTER TO ARRAY [untagged] OF SHORTCHAR;
    ptrs-: POINTER TO ARRAY [untagged] OF @ADDR;
    imports-: POINTER TO ARRAY [untagged] OF Module;
    export-: Directory;    (* exported objects (name sorted) *)
#IF @BB17
    name-: Utf8Name
#ELSE
    name-: Name
#END
  END;
```

Обработка ошибок — рассмотрение путей улучшения

```
void OmtestHelloWorld_Run (void)

{

__ENTER("OmtestHelloWorld.Run");

(*OLog_String)((_CHAR*)L"Hellow,
   World", 14);

(*OLog_Ln)();

__EXIT;

}
```

Формирование цепочки вызовов процедур, соответствующей текущему состоянию стека;

Доступ к локации ошибки из TrapViewer для Ofront, LLVM, оптимизация?

Рассмотрение вариантов:

- Itanium ABI Exception Handling Specification (DWARF 4 Standard) — на основе таблиц исключений без добавления кода в основной алгоритм;

- Setjmp/Longjmp (SJLJ) based exception handling — манипулирует состоянием фрейма активации во время рантайма.

Данные варианты привносят за собой достаточно сложные механизмы.

Вложенные процедуры

- Используется подход Ofront с копированием состояния переменных фрейма активации (нет доступа).
- Желательно улучшить эффективность кода.

```
PROCEDURE SLen (x: SET; VAR str1412:
   ARRAY OF CHAR): INTEGER;

    PROCEDURE GetSLen (): INTEGER;

    BEGIN

        RETURN LEN(str1412)

    END GetSLen;

BEGIN

    RETURN GetSLen()

END SLen;
```

- Предпочтителен запрет функциональности, как в O7

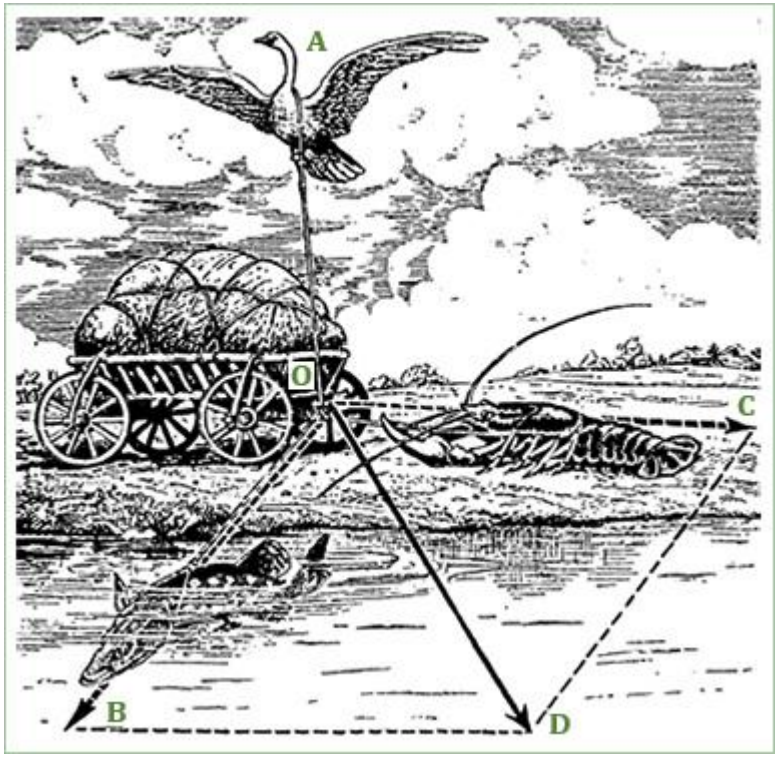
```
static INTEGER OmtestSuiteG_SLen
   (SET x, _CHAR *str1412, INTEGER
   str1412__len)

{

struct SLen__38 _s;

__ENTER("OmtestSuiteG.SLen");

_s.str1412 = (void*)str1412;
   _s.str1412__len = str1412__len;

_s.lnk = SLen__38_s;

SLen__38_s = &_s;

SLen__38_s = _s.lnk;

__EXIT;

return __39();

}
```

О совместно-независимой разработке

Совместно-независимая разработка возможна, если разработчики не мешают друг другу.

На рисунке из учебника Я.И.Перельмана Лебедь, Рак и Щука прилагают усилия во взаимно-ортогональных направлениях.

Рекомендуемая практика Carrier-Rider-Mapper как раз и сделана для того, чтобы расширяться в разных направлениях.



В МультиОберон инкорпорированы DevCompiler, Ofront, Dev2Linker, ...

Вопросы по докладу …

Дагаев Дмитрий Викторович,
Главный Эксперт,
АО «Русатом Автоматизированные системы управления»
Консультант проекта Информатика-21
forum.oberoncore.ru
www.inr.ac.ru/~info21/

dvdagaev@mail.ru