Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 10 № 1

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2020

Тематика номера:
Инструментальные средства для операционных систем реального времени, математическое моделирование и визуализация, моделирование физических процессов

Журнал публикует оригинальные статьи по следующим областям исследований: математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:

Tools for real-time operating systems, math modeling and visualization, modeling of physical processes

The Journal publishes novel articles on the following research arias: mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies,
micro- and nanoelectronics, mathematical researches and problems of numerical analysis,
history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

Н.И. Вьюкова, В.А. Галатенко, А.Н. Годунов, С.В. Самборский, И.И. Хоменков. Разделяемые библиотеки для операционной системы реального времени ……..4
В.А. Галатенко, К.А. Костюхин. Посмертный анализ процессов ОСРВ Багет с использованием аварийных дампов состояния …….17
А.В. Мальцев, М.А. Торгашев. Визуализация виртуального окружения с использованием VR-гарнитуры ……...22
Н.П. Ефимова, П.В. Крыганов, И.В. Афанаскин. Реализация методики определения предыстории при решении задачи нестационарной фильтрации...26

Разделяемые библиотеки для операционной системы реального времени

Н.И. Вьюкова[1], В.А. Галатенко[2], А.Н. Годунов[3], С.В. Самборский[4], И.И. Хоменков[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, niva@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, galat@niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nkag@niisi.ras.ru

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, sambor@niisi.ras.ru

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nkigor@niisi.ras.ru

Аннотация. В статье представлена реализация разделяемых библиотек для семейства операционных систем реального времени Багет 3.x (ОСРВ Багет 3.x). Рассматриваются методы обеспечения основных требований к разделяемым библиотекам. В частности, использование разделяемых библиотек не должно приводить к непредсказуемым задержкам во время выполнения приложений или к потере производительности, а также требовать значительной доработки ОСРВ Багет или изменений прикладного бинарного интерфейса. Приводится пример создания разделяемой библиотеки функций стандарта POSIX и ее использования в прикладной программе для семейства ОСРВ Багет 3.x.

Ключевые слова: операционная система реального времени, разделяемая библиотека

1. Введение

ОСРВ Багет 3.x – семейство операционных систем реального времени, разработанное в НИИСИ РАН для ЭВМ серии «Багет» [1]. В отличие от операционных систем общего назначения (таких как MS Windows или ОС семейства Linux), ОСРВ предназначены для создания приложений, обрабатывающих внешние события. Главная сфера их применения – управление различными приборами, работающими, в том числе, в автоматическом режиме.

В 2002 году была выпущена первая версия операционной системы семейства ОСРВ Багет 2.x. Она работала на микропроцессорах архитектуры MIPS: R3000, Комдив32, RM7000, Комдив64, Комдив128, а также Intel i486. В ОСРВ семейства Багет 2.x поддерживалась так называемая изолированная архитектура, подразумевающая, что один процессор управляет работой одного конечного устройства. Прикладная программа, работающая под управлением ОСРВ семейства Багет 2.x, представляла собой один процесс, в рамках которого могло выполняться несколько потоков управления. Такое решение позволяет сократить необходимый для работы системы объем памяти и улучшить временные характеристики. Существенным ограничением ОСРВ семейства Багет 2.x является то, что в них может выполняться только один процесс, и в случае сбоя необходима перезагрузка всего модуля.

С появлением более мощных процессоров стало возможным создание систем, в которых на одном процессоре выполняется несколько подсистем управления, разделяющих между собой вычислительные ресурсы. Для обеспечения надежности таких систем, необходимо чтобы отдельные их подсистемы были изолированы и могли при необходимости перезагружаться независимо друг от друга, без перезагрузки всего модуля. В самолетостроении такой подход получил название интегрированной модульной авионики (ИМА) [2]. Интерфейс прикладных программ для систем ИМА должен соответствовать спецификациям ARINC 653 [3].

ОСРВ семейства Багет 3.x предназначены для микропроцессоров архитектуры Комдив [4] последних поколений, располагающих существенно большими вычислительными ресурсами по сравнению с ранними моделями. Операционные системы семейства Багет 3.x, в отличие от ОСРВ семейства Багет 2.x, разрабатывались с учетом упомянутых выше требований. В частности, в них поддерживаются спецификации ARINC 653, защита памяти, планирование процессов в соответствии с расписанием, расширенные возможности обработки ошибок. При разработке ОСРВ семейства Багет 3.x использовался международный стандарт POSIX 1003.1 [5], описывающий интерфейс между операционной системой и прикладной программой.

Спецификация ARINC 653 допускает как

пользовательские, так и системные процессы. Под управлением ОСРВ семейства Багет 3.x могут выполняться ARINC-процессы (пользовательские разделы в терминологии ARINC 653) и POSIX-процессы (системные разделы в терминологии ARINC 653). Главный системный процесс (ядро) выполняется в привилегированном режиме работы процессора и выполняет функции операционной системы. Остальные процессы выполняются в пользовательском режиме.

Создание приложений для ОСРВ семейства Багет 3.x с большим числом процессов может вызвать затруднения из-за дефицита оперативной памяти. Выполняемый файл каждого процесса создается как статически скомпонованный выполняемый модуль. Программы и данные всех процессов отображаются в виртуальную память, однако в конечном счете все они должны быть размещены в физической оперативной памяти. Подкачка страниц в ОСРВ семейства Багет 3.x не предусмотрена, поскольку это противоречит принципам систем реального времени. К тому же аппаратные модули, на которых она работает, не имеют дисков.

Хотя объем оперативной памяти в аппаратных модулях на базе современных процессоров КОМДИВ довольно велик, при увеличении числа процессов может оказаться, что памяти недостаточно. Стандартный способ решения подобных проблем – разделяемые библиотеки. Разделяемая библиотека загружается в память в одном экземпляре и может использоваться одновременно несколькими процессами.

Разделяемые библиотеки для современных операционных систем имеют позиционно независимый код, менее эффективный, чем обычный. Кроме того, реализация обычных разделяемых библиотек несовместима с оптимизацией доступа к малым объектам в памяти, существенной для процессоров архитектуры MIPS. Таким образом, использование подобного подхода ведет к снижению производительности. К тому же потребовалась бы значительная доработка ОСРВ Багет для поддержки динамической загрузки разделяемых библиотек.

Поэтому была поставлена задача реализовать для ОСРВ семейства Багет 3.x разделяемые библиотеки, настроенные на фиксированные адреса и допускающие статическую компоновку прикладных выполняемых модулей на стадии их сборки.

Дальнейшее содержание статьи построено по следующему плану. В разделе 2 рассмотрена организация разделяемой библиотеки для ОСРВ семейства Багет 3.x и процедура ее построения. В разделе 3 представлен пример конфигурирования и сборки приложения, использующего разделяемые библиотеки и показаны преимущества выбранного подхода. В заключении обсуждаются ограничения, связанные с использованием разделяемых библиотек ОСРВ семейства Багет 3.x, и представлены направления дальнейших разработок.

2. Создание разделяемых библиотек ОСРВ Багет

В этом разделе рассмотрен способ описания и построения разделяемых библиотек для ОСРВ семейства Багет 3.x в соответствии с принципами, сформулированными во введении: разделяемая библиотека должна быть настроена на фиксированный адрес и допускать статическую компоновку использующих ее выполняемых файлов приложения ОСРВ Багет.

Идея реализации заключается в том, чтобы формировать разделяемую библиотеку при помощи компоновщика ld как обычный выполняемый файл формата ELF. На стадии загрузки приложения программный сегмент этого файла размещается в сегменте памяти, который отображается в адресном пространстве всех процессов, составляющих приложение ОСРВ Багет. Важное следствие, вытекающее из этого подхода, заключается в том, что множество объектных модулей разделяемой библиотеки должно быть замкнуто по ссылкам, то есть при ее компоновке должны быть успешно разрешены все ссылки на внешние символы.

Помимо этого требуются средства, позволяющие разрешать ссылки из прикладного выполняемого файла на глобальные имена, определенные в разделяемой библиотеке. Необходим также механизм, обеспечивающий корректную работу с данными разделяемой библиотеки, доступными на чтение и запись (read write – RW). Эти вопросы подробно рассмотрены в п. 2.3, п. 2.4 и в разделе 3.

В последующих пунктах данного раздела

- представлен формат описания исходных данных для генерации разделяемой библиотеки;
- приведен пример описания разделяемой библиотеки функций стандарта POSIX для ОСРВ Багет;
- рассмотрена организация готовой разделяемой библиотеки;
- описан метод ее построения из заданных исходных данных.

2.1. Исходные данные для создания разделяемой библиотеки

Для создания каждой разделяемой библиотеки мы будем использовать отдельный подкаталог, содержащий файл `shlib.def` с описанием разделяемой библиотеки, а также `Makefile` и скрипты, применяемые для сборки библиоте-

ки, которые подробно рассмотрены в п. 2.4.

Исходными данными для создания разделяемой библиотеки могут служить существующие статические библиотеки и объектные модули. Например, это могут быть библиотеки стандартных функций, поставляемые с ОСРВ Багет. Для описания состава библиотеки используются следующие make-переменные, определения которых помещаются в файл `shlib.def`.

`INCLUDED_LIBS` – список статических библиотек, все (или почти все) объектные файлы которых должны быть включены в разделяемую библиотеку.

`EXCLUDED_LIB_OBJS` – объектные файлы из библиотек списка `INCLUDED_LIBS`, которые не нужно включать в разделяемую библиотеку. Зачем может понадобиться исключать отдельные модули из состава разделяемой библиотеки? Исходные библиотеки из списка `INCLUDED_LIBS` могут содержать стартовые файлы, ссылающиеся на функцию `main`, или служебные модули, ссылающиеся на имена, которые по каким-либо причинам не могут быть определены на стадии сборки разделяемой библиотеки.

`INCLUDED_OBJS` – список объектных файлов, которые нужно дополнительно включить в разделяемую библиотеку или использовать как файлы, задающие список имен функций для включения в библиотеку. В последнем случае объектный файл может быть получен из программы на языке Си, содержащей вызовы функций, которые должны быть включены в библиотеку. Либо это могут быть объектные файлы прикладного программного обеспечения (ПО), для которого требуется создать разделяемую библиотеку.

`EXTRA_LIBS` – список дополнительных исходных библиотек, необходимых для успешной сборки разделяемой библиотеки. Список `EXTRA_LIBS` следует определить таким образом, чтобы обеспечить разрешение всех внешних ссылок, содержащихся в модулях библиотек `INCLUDED_LIBS` (за исключением модулей списка `EXCLUDED_LIB_OBJS`) и модулей `INCLUDED_OBJS`. Как правило, в этот список будут входить библиотеки ОСРВ Багет.

`EXCLUDED_OBJS` – объектные файлы из списка `INCLUDED_OBJS`, которые в конечном счете следует исключить из разделяемой библиотеки. Для чего это может понадобиться? Если в `INCLUDED_OBJS` есть объектные файлы, использованные только для того чтобы задать имена функций, которые требуется включить в разделяемую библиотеку, то такие файлы в конечном счете нужно исключить из библиотеки, указав их в списке `EXCLUDED_OBJS`.

`SHLIB` – имя разделяемой библиотеки, которая будет создана.

`SO_TEXT_ADDR` – адрес сегмента программного кода (`.text`), на который будет настроена разделяемая библиотека. Этот адрес следует выбрать таким образом, чтобы диапазон адресов кода разделяемой библиотеки не пересекался с диапазонами адресов кода и данных прикладных выполняемых и других разделяемых библиотек.

2.2. Пример описания разделяемой библиотеки

Рассмотрим в качестве примера описание исходных данных для разделяемой библиотеки, содержащей функции стандарта POSIX и предназначенной для использования в программах для POSIX-разделов приложений ОСРВ Багет. Будем считать, что операционная система установлена в каталоге, заданном make-переменной `OSBASE`. Сборка разделяемой библиотеки будет осуществляться в подкаталоге `$(OSBASE)/target/posix_shlib`. Описание исходных данных должно находиться в файле `shlib.def`, листинг которого представлен на рис. 1.

```
INCLUDED_LIBS= \
   $(OSBASE)/lib/posixR4000.a
INCLUDED_OBJS= \
   $(OSBASE)/lib/syscallsPR4000.o
EXCLUDED_LIB_OBJS= \
   p_callf.o pstartup.o istartup.o
EXCLUDED_OBJS=
EXTRA_LIBS= \
   $(OSBASE)/lib/libcR4000.a \
   $(OSBASE)/lib/libmK64.a \
   $(OSBASE)/lib/libcpuK64.a
SHLIB=posix
SO_TEXT_ADDR=0x08000000
```

Рис. 1. Листинг файла `shlib.def` с описанием разделяемой библиотеки

В ОСРВ Багет функции стандарта POSIX (предназначенные для использования в POSIX-процессах) содержатся в статической библиотеке `posixR4000.a` и в объектном файле `syscallsPR4000.o`. Эти два файла заданы как значения переменных, соответственно, `INCLUDED_LIBS` и `INCLUDED_OBJS`.

В списке `EXCLUDED_LIB_OBJS` перечислены объектные модули `p_callf.o`, `pstartup.o`, `istartup.o` из состава библиотеки `posixR4000.a`, которые не должны включаться в разделяемую библиотеку. Все они являются служебными для ОСРВ Багет и ссылаются на имена, которые не могут быть определены во время сборки разделяемой библиотеки. В частности, модуль `istartup.o` является стартовым

и ссылается на имя `main`. Модуль `pstartup.o` ссылается на `__ctors_start`, `__ctors_end` и другие имена, которые определяются в скрипте компоновки прикладных выполняемых файлов и могут зависеть от используемого скрипта.

В качестве значения переменной `EXTRA_LIBS` задан список дополнительных библиотек ОСРВ семейства Багет 3.x: `libcR4000.a`, `libmK64.a`, `libcpuK64.a`, которые будут использованы для разрешения ссылок на неопределенные имена, содержащиеся в выбранных объектных файлах. Эти библиотеки содержат функции стандартной библиотеки языка Си, а также служебные функции ОСРВ Багет.

Рассмотренное выше описание обеспечивает выполнение основного требования к составу разделяемой библиотеки – замкнутость по ссылкам на используемые имена. В общем случае для удовлетворения этого требования может потребоваться реорганизация исходных библиотек и модулей. Может потребоваться, например, разбиение некоторых модулей на части, чтобы исключить части, которые нельзя (или по каким-то причинам нежелательно) включать в разделяемую библиотеку. Такое разбиение может оказаться нетривиальным, если части модуля используют общие локальные данные или функции.

Еще одно преобразование может потребоваться, если модуль, включаемый в состав разделяемой библиотеки, содержит вызов внешней функции, которую вы не хотите вводить в разделяемую библиотеку. В этом случае непосредственный вызов внешней функции следует заменить косвенным вызовом по глобальному указателю. Инициализация указателя должна быть выполнена в неразделяемом модуле приложения, использующего данную разделяемую библиотеку, до выполнения косвенного вызова. Инициализация может быть реализована явно или через механизм конструкторов (при помощи атрибута «constructor»).

Наконец, если исходные библиотеки, используемые в определении разделяемой библиотеки, содержат одноименные модули, то необходимо выполнить переименование модулей, чтобы обеспечить уникальность имен всех задействованных объектных модулей.

2.3. Структура разделяемой библиотеки

После того как состав разделяемой библиотеки определен в соответствии с описанными выше требованиями, она может быть сгенерирована. Как упоминалось выше, разделяемая библиотека создается как обычный выполняемый файл, программный сегмент которого при загрузке приложения ОСРВ Багет помещается в оперативную память. Этот файл далее будет называться *выполняемой частью разделяемой библиотеки*. Однако помимо этого необходимы средства, позволяющие разрешить ссылки из прикладного выполняемого файла на глобальные имена, определенные в разделяемой библиотеке. Для этой цели дополнительно создается обычная, неразделяемая библиотека, содержащая модифицированные версии всех объектных файлов, использованных при сборке выполняемой части разделяемой библиотеки, а также объектный файл с сегментом RW-данных разделяемой библиотеки. Далее эта библиотека будет называться *компоновочной частью разделяемой библиотеки*, поскольку она используется при компоновке прикладных выполняемых файлов и обеспечивает разрешение ссылок на глобальные имена разделяемой библиотеки.

Таким образом, готовая разделяемая библиотека ОСРВ Багет состоит из двух файлов, имена и содержимое которых описаны ниже.

1. `$(SHLIB)_nodata.so` – выполняемая часть разделяемой библиотеки, выполняемый файл, содержащий код всех модулей, вошедших в состав разделяемой библиотеки. Файл содержит только программный код и RO-данные исходных модулей. При загрузке содержимое этого файла размещается в памяти по фиксированному адресу и отображается в адресном пространстве всех процессов, составляющих приложение ОСРВ Багет.

2. `$(SHLIB)_thin.a` – компоновочная часть разделяемой библиотеки. Содержит трансформированные объектные модули из состава `$(SHLIB)_nodata.so`, в которых удалены разделы программного кода и данных, но сохранены прочие разделы, такие как `.ctors` (конструкторы), `.dtors` (деструкторы). В таблицу символов каждого объектного модуля включены глобальные символы разделяемой библиотеки с абсолютными адресами, которые совпадают с адресами соответствующих функций или объектов данных в файле `$(SHLIB)_nodata.so`. Помимо этого в `$(SHLIB)_thin.a` добавляется объектный файл `data_segment.o`, содержащий сегмент RW-данных разделяемой библиотеки.

Таким образом, при компоновке прикладных выполняемых файлов ОСРВ Багет с использованием статического библиотечного файла `$(SHLIB)_thin.a` обеспечивается, с одной стороны, разрешение ссылок на глобальные функции и данные выполняемой части библиотеки `$(SHLIB)_nodata.so`, с другой стороны, дублирование RW-данных разделяемой библиотеки в память каждого использующего ее процесса. Подробнее процесс компоновки при-

кладного выполняемого файла с разделяемой библиотекой и организация памяти такого выполняемого файла рассмотрены в разделе 3.

2.4. Процедура построения разделяемой библиотеки

Процедура построения разделяемой библиотеки состоит из трех основных стадий:

1. Выделение замкнутого по ссылкам набора объектных файлов в соответствии с описанием разделяемой библиотеки.

2. Построение выполняемой части разделяемой библиотеки.

3. Построение компоновочной части разделяемой библиотеки.

Пусть построение разделяемой библиотеки проводится в каталоге `$(OSBASE)/target/posix_shlib`. Построение выполняется при помощи утилиты `make` под управлением файла `Makefile`, представленного на рис. 2. Далее подробно рассмотрены шаги сборки, выполняемые на каждой из стадий, со ссылками на соответствующие правила в файле `Makefile`.

1. Выделение замкнутого по ссылкам набора объектных файлов в соответствии с описанием разделяемой библиотеки реализовано как последовательность перечисленных далее шагов.

a) Получение набора объектных файлов `INCLUDED_OBJS + INCLUDED_LIBS – EXCLUDED_LIB_OBJS`. Этот шаг описывается правилом `objs0` (рис. 2). Здесь создается подкаталог `objs0_dir`, в который помещаются объектные файлы из списка `INCLUDED_OBJS` и объектные файлы всех библиотек списка `INCLUDED_LIBS` за вычетом файлов из списка `EXCLUDED_LIB_OBJS`.

b) Получение набора файлов `INCLUDED_OBJS + INCLUDED_LIBS – EXCLUDED_LIB_OBJS – EXCLUDED_OBJS` в подкаталоге `objs_dir`, см. правило `objs` на рис. 2.

c) Предварительная сборка выполняемого файла разделяемой библиотеки с целью получения его карты сборки, см. правило `$(SHLIB)0.so.map` на рис. 2.

d) Получение набора модулей из библиотек списка `EXTRA_LIBS`, которые нужны для замыкания множества объектных файлов, собранного выше в шаге *a*. Это действие описывается правилом `extra_objs` (рис. 2). Требуемый набор формируется в подкаталоге `extra_objs_dir` путем анализа карты сборки и извлечения всех упоминаемых в ней библиотечных модулей из списка `EXTRA_LIBS`.

Этот шаг завершает выполнение стадии 1. Требуемое замкнутое по ссылкам множество объектных файлов содержится в подкаталогах `objs_dir` и `extra_objs_dir`.

2. Построение выполняемой части разделяемой библиотеки состоит из следующих шагов.

a) На первом шаге строится обычный выполняемый файл `$(SHLIB).so` из замкнутого по ссылкам набора объектных файлов, построенного на предыдущей стадии, см. правило `$(SHLIB).so` на рис. 2.

Здесь, как и в шаге 1*c*, применяется скрипт компоновки `userproc.so.x` (рис. 3), согласно которому в выполняемом файле формируются два раздела: раздел `.text`, содержащий исполняемый код и RO-данные исходных объектных модулей, и раздел `.data`, содержащий RW-данные. Структура раздела `.data` в `userproc.so.x`, существенна, она будет обсуждаться подробнее далее в п. 3.4.

b) Построение выполняемой части разделяемой библиотеки, `$(SHLIB)_nodata.so`, содержащего только сегмент кода файла из `$(SHLIB).so`. Сделать это путем простого удаления раздела `.data` не удается, поскольку в этом случае, хотя раздел и удаляется, сегмент данных нулевой длины все равно остается.

Поэтому построение `$(SHLIB)_nodata.so` выполняется последовательно при помощи четырех `make`-правил (рис. 2). Сначала выделяется сегмент кода в виде бинарного файла `text_segment` и создается пустой объектный файл `empty.o`, не содержащий ни одного раздела.

Из файла `empty.o` путем встраивания в него раздела `.text`, содержимое которого берется из `text_segment`, создается файл `text_segment.o`, содержащий только сегмент кода.

Наконец, из `text_segment.o` путем компоновки со скриптом `userproc.so.x` (рис. 3) создается выполняемая часть разделяемой библиотеки, файл `$(SHLIB)_nodata.so`, содержащий только программный код и RO-данные.

3. Построение компоновочной части разделяемой библиотеки, `$(SHLIB)_thin.a` реализуется в виде последовательности описанных далее шагов.

a) Сначала создаются файлы `data_segment` и `data_segment.o` аналогично тому, как выше в шаге 2*b* были созданы файлы `text_segment` и `text_segment.o`. Здесь `data_segment` – сегмент данных выполняемого файла `$(SHLIB).so` в формате бинарного файла, `data_segment.o` – объектный файл, содержащий только раздел `.shared_data`, содержимое которого берется из `data_segment`.

```
OSBASE=/home/osuser/oc3000/oc3000-3.52.420
include shlib.def
```

```
LD=bt23i-ld
OBJCOPY=bt23i-objcopy
.PHONY: all objs0 objs clean cleanall

all: $(SHLIB).so $(SHLIB)_nodata.so $(SHLIB)_thin.a userproc_shared.x
$(SHLIB)_thin.o

objs0: $(INCLUDED_OBJS) $(INCLUDED_LIBS)
       rm -rf objs0_dir
       mkdir objs0_dir
       for f in $(INCLUDED_OBJS); do echo Copy $$f; cp $$f objs0_dir; done
       cd objs0_dir; for l in $(INCLUDED_LIBS); \
         do echo Extract all from $$l; ar x $$l; done
       cd objs0_dir; rm -f $(EXCLUDED_LIB_OBJS)

$(SHLIB)0.so.map: userproc.so.x objs0 $(EXTRA_LIBS)
       LC_ALL=C $(LD) -Map $@   -T userproc.so.x -Ttext=$(SO_TEXT_ADDR)\
         -o $(SHLIB)0.so  objs0_dir/* --start-group $(EXTRA_LIBS) --end-group

objs: objs0
       rm -rf objs_dir
       cp -r objs0_dir objs_dir
       cd objs_dir; rm -f $(EXCLUDED_OBJS)

extra_objs: $(SHLIB)0.so.map userproc.so.x objs $(EXTRA_LIBS)
       rm -rf extra_objs_dir
       mkdir -p extra_objs_dir
       egrep "^[^[:space:]]*[.]a[(]" $(SHLIB)0.so.map \
         |while IFS="()" read a b c d e; \
         do cd extra_objs_dir; echo Extract $$b from $$a; ar x $$a $$b; cd ..;\
         done

$(SHLIB).so: $(SHLIB)0.so.map userproc.so.x objs extra_objs $(EXTRA_LIBS)
       LC_ALL=C $(LD) -Map $@.map  -T userproc.so.x -Ttext=$(SO_TEXT_ADDR)\
         -o $@  objs_dir/*  extra_objs_dir/*

$(SHLIB)_thin.a: add_symbols.sh $(SHLIB).so userproc_del.so.x objs \
         data_segment.o
       rm -f $@
       rm -rf thin_objs_dir
       mkdir -p thin_objs_dir
       ./add_symbols.sh  -l $(LD) -s $(SHLIB).so -u __shared_data \
         -o thin_objs_dir -x userproc_del.so.x objs_dir/*
       ./add_symbols.sh  -l $(LD) -s $(SHLIB).so -u __shared_data \
         -o thin_objs_dir -x userproc_del.so.x extra_objs_dir/*
       ar cr $@  data_segment.o thin_objs_dir/*
       ar tv $@

data_segment:  $(SHLIB).so
       $(OBJCOPY) -v --output-target binary --only-section .data $(SHLIB).so $@

text_segment:  $(SHLIB).so
       $(OBJCOPY) -v --output-target binary --only-section .text $(SHLIB).so $@

empty.o: objs $(SHLIB).so
       $(LD)  -r -o some_obj.o objs_dir/*  extra_objs_dir/*
       $(OBJCOPY) -v --only-section .absent_section3476576 some_obj.o $@

data_segment.o:  data_segment $(SHLIB).so empty.o
       rm -f $@
       $(OBJCOPY) -v --add-section .shared_data=data_segment \
         empty.o no_symbol.o
       nm -g $(SHLIB).so |grep  "D _gp$$"|(read a b c d; $(LD) \
         -r --defsym=_gp=0x$$a --defsym=__shared_data=0xffffffff \
         -o $@ no_symbol.o)
```

```
	readelf -a $@

text_segment.o:  text_segment  empty.o
	rm -f $@
	$(OBJCOPY) -v  --add-section .text=text_segment  empty.o $@
	readelf -a $@

$(SHLIB)_nodata.so: text_segment.o userproc.so.x
	$(LD) -T userproc.so.x -Ttext=$(SO_TEXT_ADDR) -o $@ text_segment.o

$(SHLIB)_thin.o: $(SHLIB)_thin.a
	$(LD)  -r -o $@  --whole-archive $(SHLIB)_thin.a

clean:
	rm -rf  extra_objs_dir
	rm -f $(SHLIB)0.so *.map
	rm -rf thin_objs_dir
	rm -f *.bu .*.bu
	rm -f data_segment text_segment empty.o tmptmp.o no_symbol.o some_obj.o
	rm -rf objs0_dir objs_dir
	rm -f $(SHLIB)0.so.map data_segment.o text_segment.o

cleanall: clean
	rm -f $(SHLIB).so $(SHLIB)_thin.a $(SHLIB)_nodata.so $(SHLIB)_thin.o
```

Рис. 2. Файл `Makefile` для построения разделяемой библиотеки

На самом деле данные раздела `.shared_data` не являются разделяемыми. Поскольку это RW-данные, то каждый процесс, использующий разделяемую библиотеку, должен иметь свой набор этих данных.

В `data_segment.o` также определяются два символа:

- `__shared_data=0xffffffff`. Значение символа `__shared_data` не важно; этот символ нужен лишь для того, чтобы обеспечить подключение модуля `data_segment.o` при компоновке прикладных выполняемых модулей, использующих данную разделяемую библиотеку.
- `_gp` – значение, относительно которого адресуются малые данные при использовании оптимизации -G (подробнее см. п. 3.4). Значение `_gp` здесь определяется из таблицы символов файла `$(SHLIB).so`.

b) Создание библиотеки `$(SHLIB)_thin.a`. В библиотеку включается модуль с данными (`data_segment.o`) плюс тот же список модулей, которые были использованы для сборки `$(SHLIB).so` (подкаталоги `objs_dir`, `extra_objs_dir`, построенные на стадии 1). Но в данном случае используются трансформированные версии исходных модулей: в каждом из них исключено все, что вошло в разделяемую библиотеку `$(SHLIB).so`. Это достигается применением инкрементальной компоновки со скриптом `userproc_del.so.x` (рис. 4). Конструкции `/DISCARD/` (“удалить”) этого скрипта содержат ровно те разделы кода и данных, которые указаны в конструкциях `.text` и `.data` скрипта `userproc.so.x` (рис. 3)

В каждый трансформированный модуль также добавлены глобальные символы, соответствующие символам из разделов кода и данных исходных модулей; но эти символы трансформируются в абсолютные, с теми адресами, которые они имеют в файле `$(SHLIB).so`. Кроме того, в каждый модуль добавлен неопределенный символ `__shared_data`, чтобы при компоновке пользовательского приложения обеспечить подключение модуля с данными `data_segment.o`.

Все действия по трансформации модулей из подкаталогов `objs_dir`, `extra_objs_dir` для библиотеки `$(SHLIB)_thin.a` описаны в bash-скрипте `add_symbols.sh`, текст которого здесь не приводится. Аргументы `add_symbols.sh`: `-l` задает имя компоновщика, применяемого для трансформации объектных модулей; `-s` задает имя выполняемого файла, из которого берутся адреса глобальных символов, `-o` задает имя каталога для сохранения трансформированных модулей (здесь `thin_objs_dir`), `-u` задает имя символа, который должен добавляться в таблицы символов трансформированных модулей как неопределенный; `-x` задает скрипт компоновки. В конце списка аргументов перечисляются объектные файлы, которые должны быть трансформированы.

```
SECTIONS
{
  .text  0x08000000    :
```

```
  {
    *(.text.start)
    *(.text)
    *(.text.*)
    *(.k128.text)
       *(.MIPS*)
    *(.note.gnu.build-id)
    *(.rodata)
    *(.rodata.*)
  }
  .data  0x10000000    :
  {
    *(.data)
    *(.data.*)
    *(COMMON)
    . = ALIGN(32);
    *(.bss)
    *(.bss.*)
    _gp = ALIGN(16) + 0x7ff0;
    *(.sdata .lit8 .lit4)
    *(.sbss)
    *(.scommon)
  }
  /DISCARD/ :
  {
    *(.cxx_init_section)
    *(.ctors) *(.dtors) *(.ctors*) \
      *(.dtors*) *(.ctors.*) *(.dtors.*)
    *(.init) *(.fini) *(.init*) *(.fini*) \
      *(.init.*) *(.fini.*)
    *(.rel.*)
    *(.eh_frame_zeros)
    *(.dynamic .got.plt .hm_callbacks )
    *(.gcc_except_table)
    *(.gcc_except_table.*)
    *(.eh_frame)
  }
}
```

Рис. 3. Скрипт компоновки `userproc.so.x`

Сравним для наглядности выдачи утилиты `nm` для исходного модуля `objs_dir/printf.o` и трансформированного варианта этого модуля:

```
$ nm objs_dir/printf.o
00000058 T fprintf
00000090 T _fprintf
00000000 T printf
         U threadGetStdF
         U vfprintf
         U _vfprintf
$ nm thin_objs_dir/printf.o
0800d668 A fprintf
0800d6a0 A _fprintf
0800d610 A printf
         U __shared_data
         U threadGetStdF
         U vfprintf
         U _vfprintf
```

В трансформированном модуле вместо символов раздела `.text` (`T`) присутствуют абсолютные символы (`A`) и добавлен неопределенный символ `__shared_data`.

При помощи команды `objdump` нетрудно также убедиться в том, что в трансформированном модуле отсутствуют разделы кода и данных.

В Makefile (рис. 2) определено также правило сборки объектного модуля `$(SHLIB)_thin.o`, объединяющего все модули библиотеки `$(SHLIB)_thin.a`. Как показано в следующем разделе, этот модуль может быть целесообразно использовать вместо библиотеки `$(SHLIB)_thin.a` при сборке прикладных выполняемых файлов. Подробнее вопросы сборки выполняемых файлов, использующих разделяемую библиотеку, обсуждаются в разделе 3.

```
SECTIONS
{
  /DISCARD/ :
  {
    *(.text.start)
    *(.text)
    *(.text.*)
    *(.k128.text)
       *(.MIPS*)
    *(.note.gnu.build-id)
    *(.rodata)
    *(.rodata.*)
  }
  /DISCARD/ :
  {
    *(.data)
    *(.data.*)
    *(COMMON)
    *(.bss)
    *(.bss.*)
    *(.sdata .lit8 .lit4)
    *(.sbss)
    *(.scommon)
  }
}
```

Рис. 4. Скрипт компоновки `userproc_del.x`

3. Сборка приложений ОСРВ Багет, использующих разделяемые библиотеки

3.1. Сборка выполняемого модуля

Рассмотрим сборку выполняемого модуля в posix-разделе приложения ОСРВ Багет, использующего рассмотренную выше разделяемую библиотеку posix. Исходный код приложения содержится в модуле `./posix/main.c` (рис. 5).

Для сборки выполняемого модуля используется файл `./posix/makefile`, текст которого приведён на рис. 6. Файл `makefile` предусматривает сборку трех перечисленных далее вариантов выполняемого модуля.

```
#include <stdio.h>
#include <stdlib.h>

void out(void) {
  printf("Posix stopped\n");
}
```

```
int main() {{
  printf("Posix started\n");
  atexit(out);
  return 0;
}
```

Рис. 5. Прикладная программа posix-раздела

– `userproc.o` – модуль, собранный обычным способом без использования разделяемых библиотек, его сборка определяется правилом, находящимся во включаемом файле `$(OSBASE)/lib/makeuser.def`;

– `userproc_thin.o` – модуль, использующий разделяемую библиотеку посредством компоновочной части, оформленной в виде библиотеки `$(SHLIB)_thin.a`;

– `userproc_thino.o` – модуль, использующий разделяемую библиотеку посредством компоновочной части, оформленной в виде объектного файла `$(SHLIB)_thin.o`.

Правила для сборки последних двух выполняемых файлов схожи с обычным правилом, применяемым для сборки `userproc.o`, но имеют следующие отличия:

```
include ../OSBASE
include usermake.def
SHLIB=posix
ADDED_CFLAGS += -D__POSIX__
all: copy userproc.o userproc_thin.o userproc_thino.o
include $(OSBASE)/lib/makeuser.def
userclean:
	rm -f userproc.o userproc_thin.o userproc_thino.o posix* *.x

userproc_thin.o:  main.o  userproc_shared.x  posix_thin.a
	$(CC)  -nostartfiles -T userproc_shared.x -u userThreadStub -o $@ \
	main.o -Xlinker -'(' posix_thin.a $(OSBASE)/lib/libcpuK64.a \
	$(OSBASE)/lib/libmK64.a $(OSBASE)/lib/posixR4000.a \
	$(OSBASE)/lib/oc3000aR4000.a $(OSBASE)/lib/libcR4000.a -lgcc \
	-Xlinker -')' -Wl,-Map=userproc_thin.map

userproc_thino.o:  main.o  userproc_shared.x  posix_thin.o
	$(CC)  -nostartfiles -T userproc_shared.x -u userThreadStub -o $@ \
	main.o posix_thin.o -Xlinker -'(' $(OSBASE)/lib/libcpuK64.a \
	$(OSBASE)/lib/libmK64.a  $(OSBASE)/lib/posixR4000.a \
	$(OSBASE)/lib/oc3000aR4000.a $(OSBASE)/lib/libcR4000.a -lgcc \
	-Xlinker -')' -Wl,-Map=userproc_thino.map

posix.so posix_nodata.so posix_thin.a userproc_shared.x posix_thin.o:%:
../../posix_shlib/%
	cp $< $@

copy: posix.so posix_nodata.so posix_thin.a userproc_shared.x posix_thin.o
	cp posix_nodata.so ../so/posix_nodata.so

thin: userproc_thin.o
	cp $< userproc.o

thino: userproc_thino.o
	cp $< userproc.o
```

Рис. 6. Файл `makefile` posix-раздела приложения ОСРВ Багет

1. Используется скрипт компоновки `userproc_shared.x` (рис. 7), который копируется из каталога сборки разделяемой библиотеки. Этот скрипт обеспечивает, в том числе, структуру раздела `.data` для поддержки оптимизации доступа к малым данных, см. п. 3.4.

2. Используется компоновочная часть разделяемой библиотеки `posix_thin.a` (для `userproc_thin.o`) либо `posix_thin.o` (для `userproc_thino.o`).

3. Не используется объектный модуль `$(OSBASE)/syscallsPR4000.o`, поскольку он включен в разделяемую библиотеку (см. п. 2.2).

Скрипт `userproc_shared.x` и разделяемая библиотека, включая оба варианта компоновочной части, копируются в текущий каталог из каталога `posix_shlib`.

Правило `copy` обеспечивает копирование

выполняемой части разделяемой библиотеки posix_nodata.so в подкаталог ../so/. Содержимое этого подкаталога будет помещено в tar-файл и включено в образ ОСРВ Багет (см. ниже п. 3.5).

Для того чтобы собрать выполняемые модули posix-раздела, нужно в подкаталоге posix/ выполнить команду make. Если хотите использовать в приложении модуль userproc_thin.o или userproc_thino.o, то затем следует выполнить, соответственно, make thin или make thino.

3.2. Сравнение трех способов сборки выполняемых модулей

Сравним при помощи команды bt23i-objdump -h -j .text user*.o размер раздела кода (.text) в выполняемых модулях, полученных разными способами.

```
userproc.o        0x21f88=13914410
userproc_thin.o    0x2458=  930410
userproc_thino.o   0x23c8=  916010
```

Размер кода выполняемых модулей userproc_thin.o и userproc_thino.o, использующих разделяемую библиотеку, значительно меньше, чем у модуля userproc.o, собранного обычным образом.

Разница между размером кода в файлах userproc_thin.o и userproc_thino.o объясняется тем, что в первом из них подключается модуль $(OSBASE)/lib/posixR4000.a(open.o) по ссылке на имя open из модуля $(OSBASE)/lib/posixR4000.a(pstartup.o). Это нетрудно установить из содержимого карт сборки данных модулей. Напомним, что модуль pstartup.o не входит в разделяемую библиотеку (см. рис. 1). В результате при компоновке userproc_thin.o приходится подключать файл open.o из библиотеки posixR4000.a, хотя он присутствует в разделяемой библиотеке.

Таким образом, компоновка с объектным модулем posix_thin.o оказывается предпочтительнее, чем компоновка с библиотекой posix_thin.a, поскольку posix_thin.o обеспечивает разрешение ссылок на все глобальные имена определенные в разделяемой библиотеке.

На самом деле корректную компоновку можно обеспечить и при помощи библиотеки $(SHLIB)_thin.a, если включить в нее объектные модули из списка EXCLUDED_LIB_OBJS.

3.3. Включение данных разделяемой библиотеки в прикладной выполняемый модуль

Обычно при работе с данными разделяемых библиотек применяется технология «клонирования по первой записи» (clone on write – COW). Это означает, что вначале процесс работает с данными разделяемой библиотеки как с общими для всех, но как только процесс пытается выполнить запись в какую-либо страницу данных, создается индивидуальная копия страницы для данного процесса, и далее он работает уже с этой копией. Но использование подобной технологии в ОСРВ Багет могло бы привести к задержкам либо, что хуже, к исчерпанию памяти во время выполнения.

Можно было бы при старте системы выделять для каждого процесса отдельную страницу под данные разделяемой библиотеки, но это было бы не экономно, так как размер страниц памяти ОСРВ Багет довольно велик, а фактический размер RW-данных в библиотеке обычно небольшой. К тому же при таком подходе нельзя было бы поддержать оптимизацию доступа к малым объектам данных в библиотеке, которая существенна для архитектуры MIPS.

Поэтому предпочтительнее оставлять RW-данные в компоновочной части разделяемой библиотеки и включать их в прикладные выполняемые файлы на стадии компоновки. Таким образом, каждый прикладной выполняемый модуль получает свой экземпляр данных разделяемой библиотеки без каких-либо специальных действий со стороны операционной системы.

Во время сборки разделяемой библиотеки формируется объектный файл с ее данными data_segment.o, включая заполненный нулями раздел .bss. Поскольку в ОСРВ Багет адрес RW-сегмента данных пользовательского приложения фиксирован, то именно этот адрес и используется при компоновке разделяемой библиотеки (рис. 3).

Данные разделяемой библиотеки в результате компоновки прикладного выполняемого модуля должны разместиться точно по тем же адресам, которые они получали при компоновке разделяемой библиотеки (см. адрес раздела .data в скриптах userproc.so.x на рис. 3 и userproc_shared.x на рис. 7).

3.4. Поддержка оптимизации доступа к малым данным

Обычно загрузка или запись значения в память требует двух команд. Например, загрузка значения переменной F в регистр сопроцессора плавающей арифметики $f1 может выглядеть следующим образом:

```
lui $2,%hi(F)
ldc1 $f1,%lo(F)($2)
```

Здесь сначала в общий регистр ($2) загружается старшая часть адреса %hi(F), а затем осу-

ществляется загрузка значения с использованием адресации относительно $2 со смещением, равным младшей части адреса %lo(F). При использовании ABI 64 для загрузки или записи значения требуется выполнить уже не две, а четыре команды.

```
   .data  0x10000000    :
     {
       PROVIDE (_fdata = .);
       *(.shared_data)
       *(.sdata .lit8 .lit4)
       *(.dynamic .got.plt \
         .hm_callbacks )
       *(.sbss)
       *(.scommon)
       *(.data)
       *(.data.*)
       PROVIDE
(__gcc_except_table_start = .);
       *(.gcc_except_table)
       *(.gcc_except_table.*)
       PROVIDE (_edata = .);
     }
     . = ALIGN(32);
     .bss      :
     {
       PROVIDE (_fbss = .);
       *(COMMON)
       *(.eh_frame)
       . = ALIGN(32);
       *(.bss)
       *(.bss.*)
       PROVIDE (_end = .);
     }
```

Рис. 7. Фрагмент скрипта компоновки userproc_shared.x, описывающий разделы данных (.data, .bss) прикладного выполняемого модуля, использующего разделяемую библиотеку

Ключ -G*n* позволяет реализовать доступ к данным размером до *n* байт одной командой, используя адресацию относительно выделенного регистра $28 с соответствующим смещением:

```
ldc1 $f1,%gp_rel(F)($28)
```

Для того чтобы эта оптимизация была возможна, малые данные должны размещаться в памяти последовательно и их общий размер не должен превышать 64К.

Чтобы поддержать оптимизацию доступа к малым данным в приложениях, использующих разделяемые библиотеки, применяется схема размещения данных, представленная на рис. 8.

Скрипт userproc_shared.x (рис. 7) отличается от стандартного тем, что в начало выходного раздела .data добавляется содержимое раздела .shared_data, содержащего данные разделяемой библиотеки в нужном порядке (см. структуру раздела .data на рис. 3 и правило для создания файла data_segment.o на рис. 2). Малые данные разделяемой библиотеки помещаются после остальных её данных в конце раздела .shared_data. Малые данные прикладной программы, включая разделы .sbss, .scommon, помещаются сразу после малых данных разделяемой библиотеки, а остальные её данные помещаются в конце.

Рис. 8. Структура раздела .data в выполняемом файле, использующем разделяемую библиотеку

Адрес _gp, относительно которого адресуются малые данные, определяется во время генерации разделяемой библиотеки $(SHLIB).so как максимальный адрес, относительно которого возможна адресация малых данных библиотеки (см. рис. 3): *адрес_начала_малых_данных_разделяемой_библиотеки* + 32К -16. Определение символа _gp помещается в файл с данными библиотеки (см. правило data_segment.o на рис. 2).

Ограничением данного подхода является невозможность использования нескольких разделяемых библиотек в одном выполняемом модуле.

3.5. Конфигурирование приложения, использующего разделяемую библиотеку

Для того чтобы включить в образ ОСРВ Багет разделяемую библиотеку, необходимо внести следующие изменения в конфигурацию приложения.

1. В файле usermake.def в каталоге приложения добавить so/posix_nodata.so в конец строки, определяющей make-переменную TAR. В рассмотренном выше примере эта строка может выглядеть следующим образом:

```
TAR=arinc/userproc.o \
  posix/userproc.o so/posix_nodata.so
```

Тем самым разделяемая библиотека будет включена в файловую систему TAR образа ОСРВ Багет.

2. В файл `config/arinc_ag.xml` добавить строку, описывающую местонахождение выполняемой части разделяемой библиотеки:

```
<SharedLibrary
FileName="/tar/so/posix_nodata.so"
SharedLibraryId="1"
PageSize="0x10000"/>
```

4. Заключение

Представленный в работе подход к организации разделяемых библиотек позволил удовлетворить основные требования, которые предъявлялись к их реализации. Благодаря отсутствию динамической загрузки исключаются задержки на стадии выполнения приложений. Для использования разделяемых библиотек требуются минимальные дополнительные действия со стороны операционной системы: она должна обеспечить загрузку программной части разделяемой библиотеки в оперативную память и обеспечить отображение соответствующего сегмента в память всех разделов приложения. Производительность приложения, использующего разделяемые библиотеки, не снижается, поскольку сохранена возможность оптимизации доступа к малым данным.

В то же время выбранные решения привели к ряду ограничений в создании и использовании разделяемых библиотек. Некоторые из них носят принципиальный характер и не могут быть сняты в рамках выбранного подхода. Другие же могут быть преодолены за счет дальнейшего развития предложенного подхода либо не являются существенными. Ниже обсуждаются существующие на данный момент ограничения и недостатки представленной в статье реализации разделяемых библиотек для операционных систем реального времени семейства Багет 3.x.

Поддержка нескольких разделяемых библиотек. Наиболее важным ограничением является невозможность использования одновременно нескольких разделяемых библиотек в одном выполняемом модуле. Это связано со схемой компоновки данных, рассмотренной в п. 3.4. Хотя возможно наличие в приложении нескольких разделяемых библиотек, но в каждом выполняемом модуле (posix- или arinc-раздела приложения) может быть использована только одна из них. Включение нескольких разделяемых библиотек в один выполняемый модуль возможно за счет отказа от поддержки оптимизации доступа к малым данным в этих библиотеках, однако в настоящее время такая возможность не реализована.

Слабые символы. В данной реализации разделяемых библиотек не предусмотрена полноценная поддержка слабых (weak) символов. Объектные модули исходных библиотек, содержащие слабые символы, должны исключаться из состава разделяемой библиотеки. Для этого необходимо исследовать при помощи утилит `nm` или `objdump` содержимое исходных библиотек. В дальнейшем желательно реализовать автоматическую выдачу диагностики о наличии слабых символов в исходных библиотеках или объектных модулях во время сборки разделяемой библиотеки.

Разделяемые библиотеки для больших программ. Если код разделяемой библиотеки находится далеко от кода пользовательских программ (в другом 256Мб-сегменте), то требуется компилировать пользовательскую программу с ключом длинных вызовов (`-mlong-calls`). Можно реализовать сборку двух вариантов разделяемой библиотеки, с разными адресами, и использовать один либо другой из них в зависимости от размера прикладной программы.

Совпадающие имена модулей в разных исходных библиотеках. Поддержка совпадающих имен в разных исходных библиотеках затрудняется тем, что утилита `ar` не поддерживает маршрутные имена файлов в архивах. В подобных случаях сейчас требуется ручная трансформация исходных библиотек. Возможна реализация диагностики о совпадающих именах модулей в исходных библиотеках либо автоматического переименования одноименных модулей при генерации разделяемой библиотеки.

Хранение разделов неинициализированных данных. Разделы малых неинициализированных данных (`.sbss`, `.scommon`) как самих разделяемых библиотек, так и использующих эти библиотеки прикладных программ попадают в раздел `.data`, то есть хранятся в выполняемом файле. Этот недостаток реализации вряд ли можно считать существенным, поскольку размер малых данных не превышает 64Кб.

Усовершенствование компоновочной части разделяемой библиотеки. В п. 3.2 обсуждался дефект сборки выполняемых файлов с использованием библиотеки `$(SHLIB)_thin.a`. Для устранения этого дефекта достаточно реализовать включение в эту библиотеку модулей из списка `EXCLUDED_LIB_OBJS`.

Существует также ряд вопросов, которые требуют дополнительных исследований и доработок.

Использование разделяемых библиотек в других разделах приложений ОСРВ Багет. В настоящее время использование разделяемых библиотек реализовано и опробовано только в выполняемых файлах POSIX-разделов приложений ОСРВ Багет. Возможность их применения в разделах ARINC и в ядре не исследовалась. Очевидно, что для разделов ARINC по-

требуется создание своих версий разделяемых библиотек, поскольку при сборке выполняемых модулей POSIX и ARINC используются разные наборы библиотек ОСРВ Багет, и это должно учитываться при задании исходных данных для сборки разделяемых библиотек.

Поддержка разделяемых библиотек на языке C++. В настоящее время построение и использование разделяемых библиотек опробовано только для языка C. Создание библиотек, включающих программы на языке C++, может потребовать решения некоторых дополнительных задач, в частности, следующих.

- При изготовлении компоновочной части разделяемой библиотеки теряются локальные имена из разделяемого кода. Если в классе определен статический конструктор, то ссылка на него в разделе `.ctors` (который помещается в компоновочную часть разделяемой библиотеки) окажется неопределенной. Для решения этой задачи потребуется доработка процедуры генерации компоновочной части библиотеки.

- Если при сборке прикладного выполняемого файла используется компоновочная часть в виде объектного файла `$(SHLIB)_thin.o`, то выполнятся все конструкторы библиотеки, включая ненужные.

Отладка приложений, использующих разделяемые библиотеки. При сборке разделяемой библиотеки сохраняется файл `$(SHLIB).so`, содержащий относящуюся к библиотеке отладочную информацию. Он может быть загружен в дополнение к отладочной информации, относящейся к прикладному выполняемому файлу, и использован во время отладки. Однако эта схема отладки требует экспериментальной проверки.

Работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме (проекту) «38. Проблемы создания глобальных и интегрированных информационно-телекоммуникационных систем и сетей, развитие технологий и стандартов GRID. Исследование и реализация программной платформы для перспективных многоядерных процессоров (0065-2019-0002).»

Shared Libraries for Real Time Operating System

Nadezhda Vyukova, Vladimir Galatenko, Alexander Godunov, Sergei Samborskii, Igor Khomenkov

Abstract. The paper presents implementation of shared libraries for the real time operating systems of the Baget 3.x family (RTOS Baget 3.x). Methods for provision of basic requirements to the shared libraries are discussed. Namely, use of shared libraries should not cause unpredictable delays during execution or performance degradation; it should not also require significant upgrade of RTOS Baget or any kind of ABI changes. An example of building a shared library of POSIX functions and its use in an application for RTOS Baget 3.x is presented.

Keywords: real time operating system, shared library

Литература

1. А.Н. Годунов, В.А. Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы) - «Программирование», Москва, 2014, № 5, 68–76.

2. Применение единых стандартов на бортовое оборудование перспективных изделий авиационной техники. http://www.aviationunion.ru/Files/Gerasin.pdf.

3. Avionics application software standard interface, Part 1 / Required services, Aeronautical radio, Inc, March, 2007.

4. KOMDIV-64. https://ru.wikipedia.org/wiki/KOMDIV-64

5. IEEE Standard for Information Technology /Portable Operating System Interface (POSIX) Part 2: System Interface. IEEE Std 1003.1#2004. V. 2.

Посмертный анализ процессов ОСРВ Багет с использованием аварийных дампов состояния

В.А. Галатенко[1], К.А. Костюхин[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, galat@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kost@niisi.ras.ru

Аннотация. Статья содержит описание формата файлов core, используемых для посмертного (post-mortem) анализа процессов ОСРВ Багет, а также настройки и пример использования отладчика реального времени (ОРВ) для работы с этими файлами.

Ключевые слова: corefile, посмертный анализ, ОСРВ Багет, отладка, дамп памяти, ОРВ.

1. Введение

При работе с ОСРВ Багет [1], как и с любой другой операционной системой, возникают ситуации, когда тот или иной пользовательский процесс аварийно завершается. В этом случае обычные отладочные средства уже неприменимы, поскольку процесс не откликается на внешнее воздействие. В Unix-подобных системах на этот случай предусмотрено создание аварийного дампа состояния процесса, так называемого core файла.

В статье будет рассмотрен формат файла, содержащего аварийный дамп состояния пользовательского процесса ОСРВ Багет, перечислены отличия этого формата от формата файлов core ОС Линукс, а также приведен пример работы отладчика реального времени (ОРВ) с файлом core ОСРВ Багет.

В заключении сделаны выводы и выделены направления для дальнейших работ по теме посмертного анализа пользовательских процессов ОСРВ Багет.

2. Формат файла core ОСРВ Багет

Основными отечественными аппаратными платформами для запуска ОСРВ Багет являются процессоры с архитектурой mips32 и mips64.

В случае посмертного анализа в целях унификации формата данных удобно всегда оперировать 64-битными данными. Поэтому за основу формата файла core ОСРВ Багет для 32-битных архитектур был взят формат ELF32 файла core ОС Линукс для архитектуры mips64 с ABI n32 или o64 [2], а для 64-битных архитектур – формат ELF64 файла core ОС Линукс для архитектуры mips64.

Следует отметить, что основным форматом бинарных файлов, используемым как в ОСРВ Багет, так и в ОС Линукс является так называемый формат ELF (Executable and Linkable Format [3]).

Ниже в листинге 1 приведен заголовок ELF-файла core.

```
$ readelf -h core.1
ELF Header:
Magic:                     7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
Class:                     ELF32
Data:                      2's complement, big endian
Version:                   1 (current)
OS/ABI:                    UNIX - System V
ABI Version:               0
Type:                      CORE (Core file)
Machine:                   MIPS R3000
Version:                   0x1
Entry point address:       0x0
Start of program headers:  52 (bytes into file)
Start of section headers:  0 (bytes into file)
Flags:                     0x60002001, noreorder, o64, mips64
Size of this header:       52 (bytes)
Size of program headers:   32 (bytes)
```

```
Number of program headers: 4
Size of section headers:   40 (bytes)
Number of section headers: 0
```

Листинг 1. ELF заголовок файла core

Следует обратить внимание на тип файла (указан как CORE), а также на флаги, подсказывающие отладчику, каким образом следует интерпретировать данные, содержащиеся в этом файле. Нас будут интересовать флаги, определяющие формат бинарного интерфейса уровня приложения (Application Binary Interface, ABI), в данном случае это o64.

В общих чертах структура core файла приведена на рис. 1.

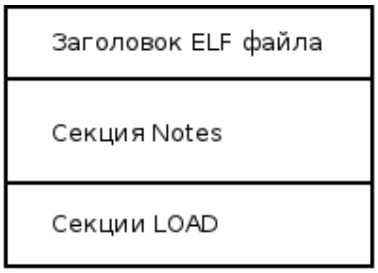


Рис. 1. Структура файла core

Заметим, что этим структура core файла не исчерпывается. Как и всякий ELF файл он может содержать дополнительные секции с данными, специфическими для целевой программно-аппаратной архитектуры. Например, информацию о состоянии используемых системных объектов ОСРВ, таких как очереди сообщений, семафоры или мьютексы.

В нашем примере программные заголовки ELF файла выглядят следующим образом (листинг 2).

```
Program Headers:
  Type        Offset    VirtAddr   PhysAddr  FileSiz MemSiz  Flg Align
  NOTE        0x0000b4 0xc001f078 0x00000000 0x006b4 0x00000     0x4
  LOAD        0x000768 0x00050d98 0x00000000 0x03c38 0x03c38 R   0x8
  LOAD        0x0043a0 0x10000000 0x00000000 0x0e940 0x0e940 RW  0x8
  LOAD        0x012ce0 0x1000e940 0x00000000 0x018ad 0x018ad RW  0x8
```

Листинг 2. Программные заголовки файла core

Секции LOAD содержат дамп памяти пользовательского процесса ОСРВ Багет, породившего файл core. Если сравнить адреса сохраненных данных с адресами загружаемых секций исходного объектного файла, то видно, что они совпадают с адресами секций .data и .bss (листинг 3).

```
$ bt23j-objdump -h posix/userproc.o

…
Разделы:
Idx Name          Size      VMA      LMA      File off  Algn
  0 .text         0005508c  00000000 00000000 00001000  2**3
                            CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         0000e940  10000000 10000000 00057000  2**3
                            CONTENTS, ALLOC, LOAD, DATA
  2 .bss          000018ad  1000e940 1000e940 00065940  2**3
                            ALLOC
…
```

Листинг 3. Программные заголовки файла core

В секции Notes содержится информация о процессе, который породил файл core: название файла, содержимое регистров, номер сигнала, заставивший процесс породить core файл.

Если взглянуть на эту секцию посредством утилиты readelf, то можно увидеть, что она состоит из нескольких записей (листинг 4).

```
$ readelf -n core.1

Notes at offset 0x000000b4 with length 0x000006b4:
  Owner                Data size  Description
  CORE                 0x000001e0 NT_PRSTATUS (prstatus structure)
  CORE                 0x00000108 NT_FPREGSET (floating point registers)
  CORE                 0x000001e0 NT_PRSTATUS (prstatus structure)
  CORE                 0x00000108 NT_FPREGSET (floating point registers)
  CORE                 0x00000080 NT_PRPSINFO (prpsinfo structure)
```

Листинг 4. Содержимое секции Notes

В данном примере секция Notes содержит записи о двух потоках управления в структурах prstatus (идентификаторы, содержимое регистров общего назначения) и в структурах fpregset (плавающие регистры), а также общую информацию о процессе, породившем файл core в структуре psinfo (имя файла-образа, аргументы). Размеры структур prstatus и psinfo фиксированы и различаются размерами в зависимости от того, является ли целевая архитектура 32- или 64-битной, а также от того, какой тип mips ABI используется. Следует отметить, что современные версии библиотеки GNU BFD (Binary File Description) не поддерживают ABI o64, поэтому авторами статьи среди прочих изменений отладчика OPB для поддержки файлов core OCPB Багет была произведена модификация используемой отладчиком библиотеки BFD. При этом стоит упомянуть о том, что с ABI n32 библиотека BFD работает корректно.

В листинге 5 приведен пример сеанса работы OPB с файлом core (комментарии авторов в листинге выделены полужирным шрифтом). Из этого листинга видно, что почти всю необходимую информацию OPB получает и отображает корректно.

```
Отладчику передается в качестве аргумента имя файла-образа и имя файла core
$ OPB40-gdb posix/userproc.o -c core.1
GNU gdb (GDB) 7.11
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=mips64-
oc2000".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from posix/userproc.o...done.

Прочитана информация о двух потоках управления
[New LWP 1]
[New LWP 2]

Информации о бинарном файле-образе в разделе psinfo нет
Core was generated by `'.
#0  main () at main.c:14
14      main.c: Нет такого файла или каталога.
[Current thread is 1 (LWP 1)]
(gdb) info threads
  Id   Target Id         Frame
* 1    LWP 1             main () at main.c:14
  2    LWP 2             0xc419bad8 in ?? ()
(gdb) info frame
```

```
Stack level 0, frame at 0x101fff68:
    pc = 0xf8 in main (main.c:14); saved pc = 0x1c60
    called by frame at 0x101f9ce0
    source language c.
    Arglist at 0x101f9ca8, args:
    Locals at 0x101f9ca8, Previous frame's sp is 0x101f9ca8
    Saved registers:
  s8 at 0x101f9c98, ra at 0x101f9ca0, pc at 0x101f9ca0
(gdb) info registers
              zero               at           v0           v1
 R0   0000000000000000 ffffffffffffffec 0000000000000000 0000000000000001
              a0               a1           a2           a3
 R4   0000000000000000 00000000cfb86e68 ffffffffc07fa060 ffffffffc051bf20
              t0               t1           t2           t3
 R8   ffffffffc051c1c0 ffffffffc05206a8 0000000000000002 0000000000000334
              t4               t5           t6           t7
 R12  0000000000000620 ffffffffc0000000 ffffffffe480fff3 ffffffffe480ffe0
              s0               s1           s2           s3
 R16  ffffffffc051c058 0000000000000000 0000000000000000 0000000000000000
              s4               s5           s6           s7
 R20  0000000000000000 0000000000000000 0000000000000000 0000000000000000
              t8               t9           k0           k1
 R24  ffffffffc0517ae8 0000000000000000 0000000000000000 00000000101fbfe0
              gp               sp           s8           ra
 R28  ffffffffc001ffa0 00000000101fff38 00000000101fff38 00000000000000f8
              sr               lo           hi           bad
      ffffffffe480fff3 000000e740834000 00000030c79baf80 0000000000000000
              cause            pc
      0000000000000000 00000000000000f8
              fsr              fir
               c07f0dc8         00000000
```

Листинг 5. Сеанс работы ОРВ с файлом core

3. Заключение

В статье описывается новая возможность посмертной отладки пользовательских процессов ОСРВ Багет. Дальнейшие работы в этом направлении могут состоять в доработке формата файлов core, включающие больше данных о породившем файл core процессе, например, имя файла-образа, а также дополнительные программные секции, содержащие информацию о системных объектах ОСРВ Багет, относящихся к данному процессу. Кроме того, рекомендуется доработать отладчик ОРВ и сервер отладки ОСРВ Багет для создания возможности порождения файлов core отлаживаемого процесса ОСРВ Багет непосредственно из отладчика ОРВ.

Работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме (проекту) «38. Проблемы создания глобальных и интегрированных информационно-телекоммуникационных систем и сетей, развитие технологий и стандартов GRID. Исследование и реализация программной платформы для перспективных многоядерных процессоров (0065-2019-0002).»

Postmortem analysis of RTOS Baget processes using core files

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The article describes the core file format used for post-mortem analysis of RTOS Baget processes, as well as settings and an example of using a real-time debugger to work with these files.

Keywords: corefile, postmortem analysis, RTOS Baget, debugging, memory dump, real-time debugger.

Литература

1. Godunov A.N., Soldatov V.A. Baget real-time operating system family (features, comparison, and future development) - Programming and Computer Software, V. 40, N 5, 2014, P. 259-264.
2. MIPS ABI History, https://www.linux-mips.org/wiki/MIPS_ABI_History
3. ELF format specification, http://www.skyfree.org/linux/references/ELF_Format.pdf

Визуализация виртуального окружения с использованием VR-гарнитуры

А.В. Мальцев[1], М.А. Торгашев[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, avmaltcev@mail.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, mtorg@mail.ru

Аннотация. В работе предлагаются решения для погружения человека в моделируемую на компьютере трехмерную среду посредством современной гарнитуры виртуальной реальности. Рассматривается подход к использованию данных от встроенной системы трекинга, а также вопросы построения стереопары для гарнитуры Oculus Rift.

Ключевые слова: визуализация, виртуальная среда, стерео, гарнитура виртуальной реальности, трекинг

1. Введение

В настоящее время виртуальная реальность (VR) является одним из актуальных и быстро развивающихся направлений в различных сферах человеческой деятельности: от компьютерных игр до сложных имитационно-тренажерных комплексов. В основе любой VR-системы лежит синтезируемая с помощью вычислительных устройств трехмерная среда, в которой настоящие сущности и явления заменяются виртуальными моделями. Обеспечение визуального восприятия этой среды человеком составляет суть процесса визуализации, или рендеринга.

Одна из важных задач при визуализации трехмерной среды в системах виртуального окружения заключается в увеличении степени погружения пользователя в эту среду. Для ее решения применяется ряд подходов, которые основаны на использовании стереоскопического рендеринга, а также реализации возможности перемещения в виртуальном пространстве. Эффект применения стерео визуализации базируется на том, что зрение человека от природы бинокулярно: два различных изображения от левого и правого глаза совмещаются мозгом в одно с получением информации о расстоянии до предметов и глубине пространства. Поэтому при рендеринге в режиме стерео каждый кадр содержит стереопару, т.е. два изображения моделируемой среды, полученных с виртуальных камер для левого и правого глаза. Каждое из этих изображений должно наблюдаться только соответствующим ему глазом пользователя. Это достигается с помощью различных методов создания и отображения стерео. Оба изображения могут выводиться на один и тот же экран (например, анаглифический, поляризационный [1] и др. подходы) и разделяться посредством специальных 3D очков [2], каждая линза которых пропускает только одно изображение из двух. В данном случае, чтобы обеспечить высокую степень погружения человека в виртуальную среду целесообразно использовать не простые плоские экраны, а установки типа CAVE ([3]). Перемещения пользователя и изменения ориентации его головы для формирования правильных изображений на экранах отслеживаются с помощью внешних систем трекинга или встроенных в очки.

Наибольшей на сегодняшний день степени погружения позволяют достичь шлемы и гарнитуры виртуальной реальности [4]. В них для каждого глаза предусмотрен независимый LCD экран (или одна из половин общего экрана). Экраны располагаются в непосредственной близости от глаз (рис. 1), а размещаемая перед ними оптика позволяет обеспечить максимальный охват поля зрения. VR-шлемы и гарнитуры также имеют встроенный трекинг головы.

В данной работе предлагаются методы и подходы для рендеринга трехмерных виртуальных сцен с использованием возможностей современных VR-гарнитур на примере устройства Oculus Rift CV1. Рассматривается задача синхронизации положения и ориентации виртуальных камер с положением и ориентацией головы пользователя, определяемым посредством встроенной системы трекинга, а также формирования стереопары, выводимой в гарнитуру. Рассмотрим их подробнее.

Рис. 1. VR-гарнитура

2. Трекинг головы пользователя

Помимо воспроизведения стереоизображений в глаза человека, используемая в данной работе VR-гарнитура Oculus Rift CV1 имеет возможность трекинга головы пользователя. Локальная система координат (СК) головы, расположенная в ее центре, задается так, что ось *Y* смотрит вверх, *X* – направо, *Z* – назад. В момент установки драйверов производится калибровка устройства с сохранением некоторого начального положения (позиции и ориентации) головы пользователя относительно внешнего сенсора, входящего в комплект гарнитуры. Обозначим соответствующую этому положению локальную СК как BCS (*basic coordinate system*, рис. 2).

Используя информацию от сенсора и встроенных датчиков (акселерометр, магнитометр, гироскоп), устройство Rift с частотой до 90 Гц вычисляет текущее смещение и ориентацию головы относительно сохраненного при калибровке начального положения. Количество подключаемых внешних сенсоров может варьироваться от одного до трех в зависимости от необходимой площади рабочей области. Доступ к рассчитанным параметрам осуществляется с помощью Oculus SDK, путем вызова функции *ovr_GetEyePoses*. Она возвращает координаты

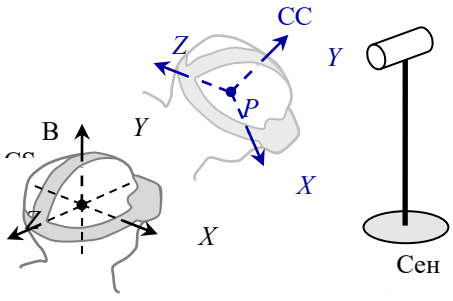


Рис. 2. Трекинг головы с гарнитурой Oculus

(*x*, *y*, *z*) начала текущей локальной СК CCS (*current coordinate system*, рис. 2) головы, представленные в системе BCS, а также нормализованный кватернион ориентации $Q = (a, b, c, d)$. На основе Q можно получить матрицу поворота M_R, трансформирующую оси СК BCS в оси СК CCS. Как показано в работе [5],

$$M_R = \begin{pmatrix} 1-2b^2-2c^2 & 2ab-2dc & 2ac+2db & 0 \\ 2ab+2dc & 1-2a^2-2c^2 & 2bc-2da & 0 \\ 2ac-2db & 2bc+2da & 1-2b^2-2a^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

При формировании изображений трехмерной сцены, видимых виртуальными камерами и транслируемых в VR-гарнитуру, необходимо обеспечить повтор этими камерами движений и поворотов головы пользователя. Для этого перед визуализацией объектов сцены производится коррекция текущей модельно-видовой матрицы M_{mv}, задающей преобразование из локальной СК OCS объекта в видовую СК VCS камеры. Коррекция выполняется по следующей формуле:

$$M'_{mv} = M_R \cdot M_T \cdot M_{mv},$$

где M_T – матрица сдвига начала системы BCS в точку $P = k \cdot (x, y, z)$, k – коэффициент масштабирования из метров в единицы измерения сцены (мм, см и т.д.),

$$M_T = \begin{pmatrix} 1 & 0 & 0 & -kx \\ 0 & 1 & 0 & -ky \\ 0 & 0 & 1 & -kz \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

3. Рендеринг изображения

Для создания стереопары в трехмерной сцене размещаются две виртуальные камеры, соответствующие левому и правому глазу. Векторы их взглядов выбираются параллельными друг другу, а оси X локальных систем VCS – сонаправленными и лежащими на одной прямой. Чтобы обеспечить правильное восприятие человеком изображений стереопары, расстояние L между камерами должно соответствовать расстоянию между его зрачками. Параметр L можно получить для конкретного пользователя из настроек гарнитуры Oculus Rift или приравнять его к среднестатистическому значению 0.064 м.

Рис. 3. Текстура для стереопары

Перед выполнением рендеринга производится смещение и поворот камер относительно их начального положения в виртуальной среде на основе данных о текущей позиции и ориентации головы пользователя, получаемых от системы трекинга Oculus Rift. Данная задача решается согласно подходу, описанному в п.2. Визуализация изображений трехмерной сцены для левого и правого глаз производится в соответствующие половины одной RGB-текстуры T, которая создается посредством вызова специальной функции *ovr_CreateTextureSwapChainGL*, входящей в Oculus SDK. Необходимая ширина W и высота H текстуры T определяются данной функцией в зависимости от используемого поля зрения (FOV) виртуальных камер. В нашем случае для каждой камеры задается рекомендованное устройством Rift значение FOV = 44°. При этом $W = 3072$, $H = 1776$ (рис. 3). Идентификатор для T можно получить с помощью функции *ovr_GetTextureSwapChainBufferGL*. Кроме текстуры T, используя упомянутые функции, требуется сгенерировать одноканальную текстуру D с аналогичными размерами, которая будет необходима для записи данных буфера глубины.

Чтобы избежать лишнего копирования данных при визуализации трехмерной виртуальной сцены в текстуру T, создается внекадровый буфер FBO (*framebuffer object*), выбираемый в качестве текущей цели рендеринга и связываемый с T и D. Изображение, получаемое от виртуальной камеры для левого глаза, записывается в область вывода, начинающуюся в точке (0, 0) и имеющую размеры $W/2$, H (рис. 3). Установка данной области производится посредством вызова функции *glViewport* (0, 0, $W/2$, H) из графической библиотеки OpenGL. Для камеры правого глаза область вывода имеет идентичные размеры, но ее начало расположено в точке ($W/2$, 0). В таком случае используется вызов *glViewport* ($W/2$, 0, $W/2$, H).

После того, как содержащая стереопару текстура T сформирована, ее необходимо передать в гарнитуру Rift путем вызова функции *ovr_SubmitFrame* или *ovr_EndFrame*. Получая изображения для левого и правого глаз, гарнитура автоматически выполняет их коррекцию для компенсации искажений, вносимых оптикой. Далее они отображаются на встроенные экраны и становятся видимы пользователю.

На рисунке 4 представлен пример стереопары (до ее коррекции гарнитурой) для устройства Oculus Rift CV1, полученной при реализации описанных подходов.

4. Заключение

В данной работе была рассмотрена задача стерео визуализации трехмерных виртуальных сцен с использованием современной гарнитуры виртуальной реальности Oculus Rift. На основе

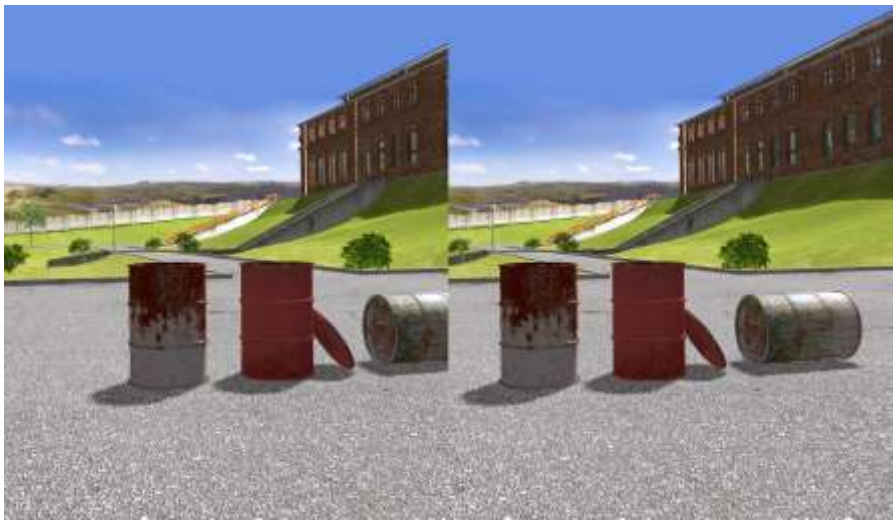


Рис. 4. Пример стереопары для Oculus Rift CV1

предложенного подхода к использованию данных от встроенной в устройство системы трекинга при синхронизации изменения видимого пользователем изображения виртуальной среды с поворотами и перемещениями его головы в реальном пространстве, а также метода синтеза соответствующей стереопары были реализованы оригинальные программные модули. Также была проведена их апробация в составе системы визуализации трехмерных виртуальных сцен, разработанной в ФГУ ФНЦ НИИСИ РАН, которая показала адекватность описанных решений и возможность их применения в системах виртуального окружения.

Публикация выполнена в рамках государственного задания по проведению фундаментальных научных исследований (ГП 14) по теме (проекту) «34.9. Системы виртуального окружения: технологии, методы и алгоритмы математического моделирования и визуализации». (0065-2019-0012).

Virtual environment visualization with using VR headset

Andrey Maltsev, Mikhail Torgashev

Abstract. The paper proposes solutions for a person immersion to computer-simulated three-dimensional environment by means of modern virtual reality headset. An approach for using data from imbedded tracking system is considered as well as questions of stereo pair creation for Oculus Rift headset.

Keywords: visualization, virtual environment, stereo, virtual reality headset, tracking

Литература

1. А.В. Мальцев, Д.В. Омельченко. Технологии и методы погружения оператора в трехмерную виртуальную среду. «Труды НИИСИ РАН», Т. 9 (2019), № 2, 59-63.

2. 3D очки и их разновидности, http://www.mir3d.ru/articles/15993/ (*дата обращения: 09.01.2020*).

3. Cave automatic virtual environment, https://en.wikipedia.org/wiki/Cave_automatic_virtual_ environment (*дата обращения: 09.01.2020*).

4. The top VR Headset - Oculus Rift, HTC Vive, PlayStation VR comparison, and more, https://www.gameskinny.com/8ka4n/the-top-vr-headset-oculus-rift-htc-vive-playstation-vr-comparison-and-more (*дата обращения: 09.01.2020*).

5. М.В. Михайлюк. Основы компьютерной графики. М., МГТУ МИРЭА, 2011.

Реализация методики определения предыстории при решении задачи нестационарной фильтрации

Н.П. Ефимова[1], П.В. Крыганов[2], И.В. Афанаскин[3]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, efinatka@gmail.com;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kryganov@mail.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ivan@afanaskin.ru

Аннотация. Рассмотрено аналитическое решение уравнения пьезопроводности для модели радиального композитного пласта с учетом суперпозиции реальных и эквивалентных дебитов. Разработана и протестирована программа для интерпретации гидродинамических исследований скважин с использованием алгоритма восстановления предыстории работы скважины для двух гидродинамических моделей.

Ключевые слова: гидродинамические исследования скважин, кривая восстановления давления, история работы скважины

1. Введение

Одним из методов определения геологического строения, энергетических параметров и фильтрационно−емкостных свойств пластов−коллекторов нефти и газа являются гидродинамические исследования скважин (ГДИС) – совокупность технологических операций по возбуждению пласта путем отбора из него пластовой жидкости или путем закачки в него жидкости и проведение соответствующих замеров дебита и давления в возмущающей и наблюдательных скважинах с последующей интерпретацией замеров. В работе [1] был проведен анализ влияния длительности учитываемого изменения дебита скважины на результаты ГДИС и предложен новый методический прием восстановления части неизвестной истории ее работы. Основная суть нового методического приема заключается в том, что режим работы скважины делится на два периода – известную историю и неизвестную предысторию, а в выражение для давления на забое скважины с учетом суперпозиции дебитов вводятся параметры, условно названные эквивалентным временем предыстории $t_э$ и эквивалентным дебитом предыстории $q_э$. Таким образом, в аналитическом решении уравнения фильтрации помимо параметров интерпретационной модели, характеризующих условия на скважине, в пласте и на границе, появляются два дополнительных параметра. Увеличение числа неизвестных параметров в модели увеличивает степень свободы при решении обратной задачи и может привести к неопределенности параметров. Кроме того, варьируя дополнительными параметрами, мы можем повлиять на характер диагностического графика Бурдэ, который и определяет интерпретационную модель. Диагностический график Бурдэ в билогарифмических координатах представляет собой семейство двух кривых, одна из которых – это изменение давления Δp во времени, а вторая – производная Δp по функции суперпозиции, учитывающей историю работы скважины [2]. По оси ординат откладывается давление и его производная, а по оси абсцисс - время.

Правомерность и целесообразность такого подхода была показана в работе [1] для простой модели - радиальной фильтрации в однородном бесконечном пласте при различных гипотетических технологических особенностях истории работы скважины. В альфа-версии специально созданного программного обеспечении (СПО) данная модель была реализована и описана в работе [1].

Практическое применение предложенного подхода зависит от возможности включения дополнительных неизвестных параметров в некоторые более сложные интерпретационные модели. С этой целью была создана и протестирована бета-версия СПО, в которую была включена еще одна гидродинамическая модель – радиальный композитный коллектор.

2. Программа для интерпретации ГДИС

Разработана бета-версия программы для интерпретации ГДИС методом наилучшего совмещения [4] при прослеживании изменения забойного давления в скважине, работающей с

произвольным дебитом, с использованием алгоритма определения предыстории [1]. В программе используются две геолого-гидродинамические модели пласта, таких как: однородный бесконечный пласт и двухзонный пласт с круговой границей раздела.

Графический интерфейс написан с использованием языка программирования C++. Экранная форма имеет стандартный вид Windows и состоит из 5-ти основных частей: полоса меню, панель инструментов, область графика, инструментальная линейка для графика и всплывающее меню. Аналитические уравнения математических моделей, численное обратное преобразование Лапласа и метод наилучшего совмещения описаны на фортране.

Импорт данных для интерпретации реализован из текстового файла и из файла данных Microsoft Excel. Исходные данные и данные интерпретации хранятся в файле – проекте в формате XML (ExtensibleMarkupLanguage).

Программа была проверена с помощью сравнительного тестирования с Saphir Kappa Engineering. Получено хорошее совмещение результатов расчетов с помощью СПО и результатов, полученных с помощью выше описанной программы.

3. Композитная модель

Радиальные композитные гидродинамические модели достаточно широко распространены в практике ГДИС. В этих случаях гидропроводность пласта меняется в горизонтальном направлении от забоя скважины, рис. 1.

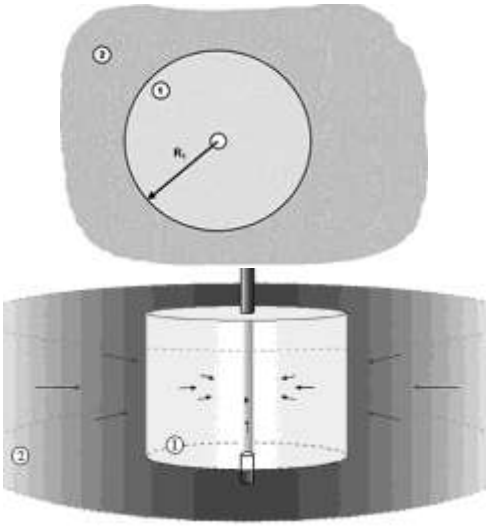


Рис. 1. Модель двухзонного пласта с круговой границей раздела и неограниченной внешней границей (радиальный композитный коллектор)

Причинами изменения гидропроводности, учитываемые при интерпретации ГДИС, могут быть: закачка флюида с резко отличающимися свойствами, изменение насыщенности из-за водоносного горизонта, эксплуатация скважины ниже давления насыщения.

Аналитическое решение, моделирующие такой кусочно-неоднородный по простиранию пласт, можно определить как двухзонный пласт с круговой границей раздела.

В этой модели возмущающая скважина радиуса r_c окружена двумя зонами с различными, но постоянными в каждой зоне фильтрационными параметрами. Гидропроводность и пьезопропроводность первой зоны (ближайшей к скважине) обозначим как ε_1 и κ_1, во второй зоне - ε_2 и κ_2. Радиус границы раздела между первой и второй зонами обозначим R_1.

В начальный момент времени во всем пласте было постоянное давление $p(\mathrm{r},0) = p_0 = const$. В момент времени $t = 0$ скважина запущена в работу с постоянным дебитом Q_0. Точное решение этой задачи в произвольный момент времени t в изображениях по Лапласу имеет вид [3]:

$$\Delta\bar{P}_1(r,s) = \frac{\bar{Q}}{s}\left[A_1\frac{I_0(r\sigma_1)}{I_0(r_1\sigma_1)} + K_0(r\sigma_1)\right], r_c \le r \le R_1 \quad , (1)$$

$$\Delta\bar{P}_2(r,s) = \frac{\bar{Q}}{s}\gamma A_2 K_0(r\sigma_2), R_1 \le r \le \infty \quad , (2)$$

где

$$A_1 = \left[\gamma K_0(r_1\sigma_2)K_0(r_1\sigma_1) - \gamma K_0(r_1\sigma_1)K_0(r_1\sigma_2)\right]/Z,$$

$$A_2 = \left[K_0(r_1\sigma_1)I_1(r_1\sigma_1) - K_1(r_1\sigma_1)I_0(r_1\sigma_1)\right]/\left[Z\cdot I_0(r_1\sigma_1)\right],$$

$$Z = \left[K_1(r_1\sigma_2)I_0(r_1\sigma_1) - \gamma K_0(r_1\sigma_2)I_1(r_1\sigma_1)\right]/I_0(r_1\sigma_1),$$

$$\bar{Q} = {Q_0\mu}/{2\pi k_1 h_1},$$

$$\sigma_i = \sqrt{s/\kappa_i}, (i = 1,2),\ \gamma = \sqrt{\varepsilon_2/\varepsilon_1},$$

$\Delta\bar{P}(r,s)$ – лапласово изображение изменения давления $\Delta p(r,t)$, s – параметр преобразования Лапласа, I_0, I_1, K_0 и K_1 – символы модифицированных функций Бесселя.

Параметры, определяемые при решении обратной задачи гидродинамики [5]: ε_1, κ_1, ε_2, κ_2, R_1.

Подставляя выражение для давления с учетом суперпозиции дебитов:

$$p(t) = p_0 - \frac{B\mu}{2\pi k_1 h_1}\left\{q_э\left[p_D\left(t_э + t_j' + \Delta t\right)_D - p_D\left(t_j' + \Delta t\right)_D\right] + \right.$$

$$+q_1'\left[p_D\left(t_j' + \Delta t\right)_D - p_D\left(t_э + t_j' - t_1' + \Delta t\right)_D\right] +$$

$$+q_2'\left[p_D\left(t_э + t_j' - t_1' + \Delta t\right)_D - p_D\left(t_э + t_j' - t_2' + \Delta t\right)_D\right] + \ldots$$

$$\left. \ldots + q_j'\left[p_D\left(t_э + t_j' - t_{(j-1)}' + \Delta t\right)_D - p_D\left(t_э + \Delta t\right)_D\right]\right\}, j = 1,2\ldots n$$

подробно описанное в работе [1] в аналитические выражения (1) и (2), мы получаем решение для модели двухзонного радиального пласта с двумя дополнительными параметрами $t_э$ и $q_э$.

4. Диагностика композитной модели

Диагностические графики Бурдэ [2] для композитной модели представлены семейством кривых, рис. 2.

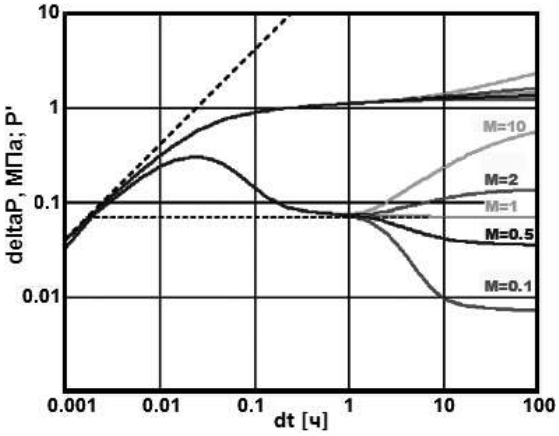


Рис. 2. Диагностические графики для композитной модели при M = 0.1, 0.5, 1, 2, 10 [2]

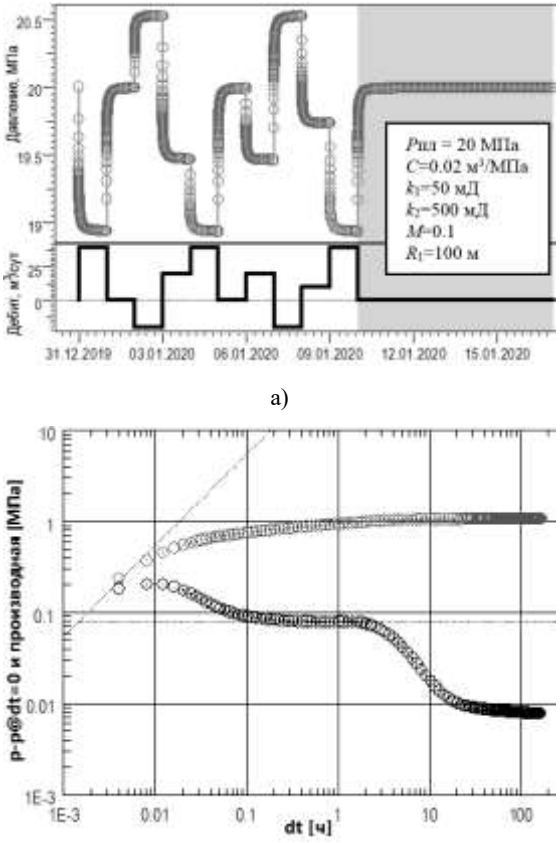
После окончания влияния ствола скважины на билогарифмическом графике производная Бурдэ выходит на горизонтальную прямую, что соответствует параметрам первой зоны фильтрации, рис. 1. Далее производная либо растет и выходит на второй горизонтальный участок, либо падает и также выходит на горизонтальный участок, в зависимости от отношения проводимостей двух зон, то есть:

$$M = \frac{(k/\mu)_1}{(k/\mu)_2}$$

Второй горизонтальный участок соответствует параметрам второй зоны зонально-неоднородного пласта. В предельных случаях, когда $M \to 0$, внешняя граница зоны – это контур постоянного давления, а при $M \to \infty$ – замкнутый контур.

5. Тестирование композитной модели с учетом и без учета дебита предыстории

Тестирование модели проводилось путем моделирования гипотетических примеров в программе Saphir Kappa Engineering и их интерпретации в СПО. Наиболее общий случай – это пример произвольного 10-ти ступенчатого изменения дебита. Была смоделирована кривая восстановления давления (КВД) с полным заданным дебитом и заданными фильтрационно-емкостными параметрами модели радиального композитного пласта, рис. 3а. Диагностика этой модели представлена на рисунке 3б.



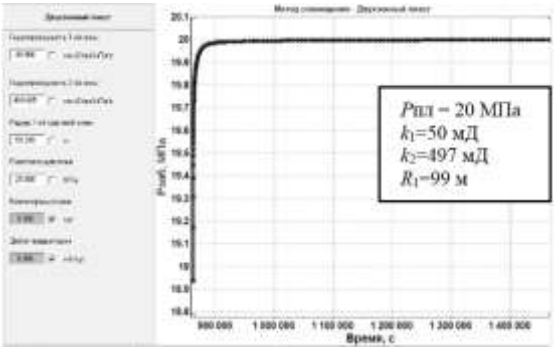
а)

б)

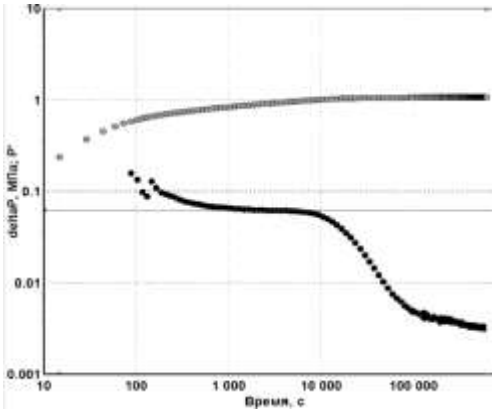
Рис. 3. Смоделированный пример в программе Saphir Kappa Engineering;

а) расчет изменения давления по заданному дебиту и заданным параметрам пласта, б) диагностический график Бурдэ

Далее гипотетическая КВД была диагностирована в СПО, рис. 4б и обработана методом наилучшего совмещения по модели друхзонного пласта с нулевыми параметрами предыстории, рис. 4а. Результаты интерпретации КВД в СПО, рис. 4а, практически совпадают с параметрами, заданными для моделирования в программе Saphir Kappa Engineering, рис. 3а.



а)

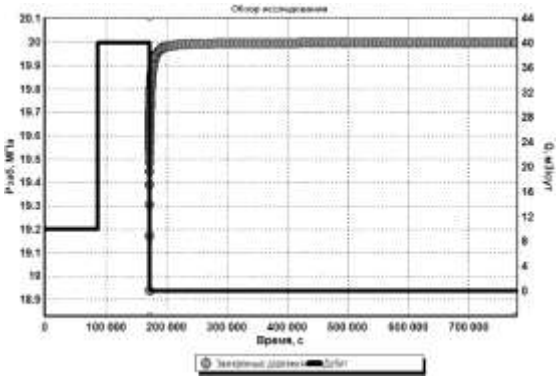


б)

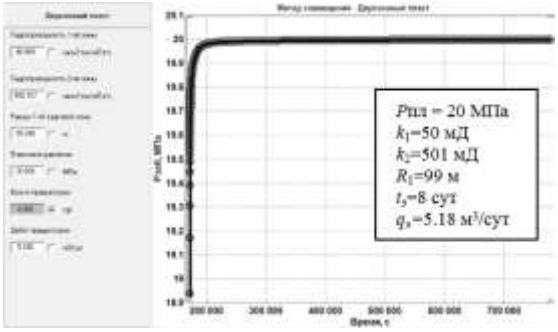
Рис. 4. Интерпретация гипотетической КВД в СПО;

а) метод наилучшего совмещения, расчитанные параметры модели, б) диагностический график Бурдэ

Для тестирования модели с дополнительными параметрами в СПО исключаем восемь из десяти участков истории работы скважины из интерпретации КВД, рис. 5а.



а)



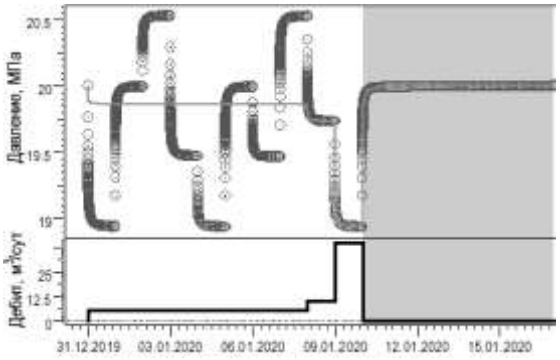
б)

Рис. 5. Интерпретация КВД с неполной историей работы скважины в СПО по модели друхзонного пласта;

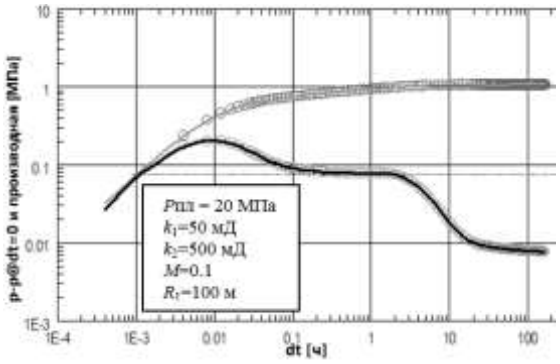
а) расчетная КВД и неполный дебит, б) метод наилучшего совмещения, расчитанные параметры модели

Интерпретируя гипотетическую кривую по модели пласта с неполной историей работы скважины находим 4 параметра модели и 2 параметра предыстории, рис. 5б.

Полученными эквивалентными параметрами заменяем отсеченную ранее предысторию, рис.6а, и проводим интерпретацию в программе Saphir Kappa Engineering, рис. 6б.



а)



б)

Рис. 6. Интерпретация КВД с воссозданной историей работы скважины в программе Saphir Kappa Engineering по модели композитного радиального пласта;

а) замена предыстории эквивалентными параметрами, б) параметры интерпретации и совмещенные кривые

Результаты обработки гипотетической КВД с восстановленной предысторией работы скважины, рис. 6а, идентичны с исходными данными для моделирования, рис. 3а, что подтверждает эффективность нового подхода и успешную работу СПО для конкретного гипотетического примера.

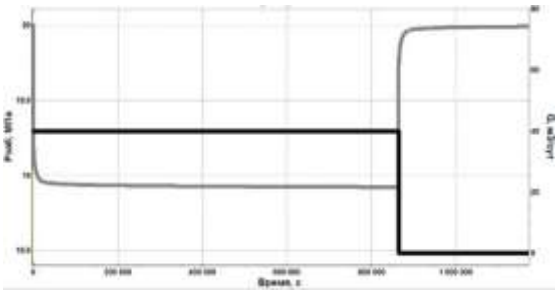
Из диагностики композитной модели, рис. 2. видно, что характер производной для композитной модели сильно отличается в зависимости от параметра M. Из этого следует предположение, что при $M<<1$, возможно, нахождение эквивалентных параметров, приведет к неопределенности их оценки. Для тестирования новой модели в СПО были рассчитаны варианты задач, когда $M<1$ и $M>1$, при различных временах ра-

боты и дебитах скважины.

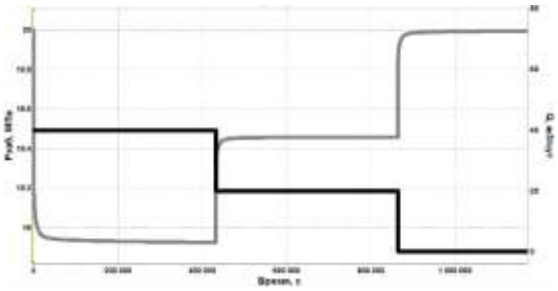
На рис. 7 представлены гипотетические примеры кривых изменения забойного давления, рассчитанных по модели радиального композитного пласта при различных технологических особенностях истории работы скважины: работа скважины с постоянным дебитом, двухступенчатый и многоступенчатый дебит, периодический и произвольный дебит.

Для этих случаев был проведен сравнительный анализ по той же схеме, что и в приведенном выше примере.

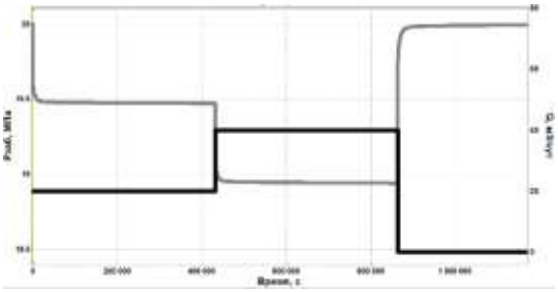
Анализ модели композитного пласта позволил сделать вывод, аналогичный тому, что был получен для модели однородного бесконечного пласта [1]. Если искомый дебит и время заданы в виде одной ступени, то с помощью СПО их можно определить точно. Для случая произвольного дебита данный подход позволяет определить эквивалентные дебит и время.



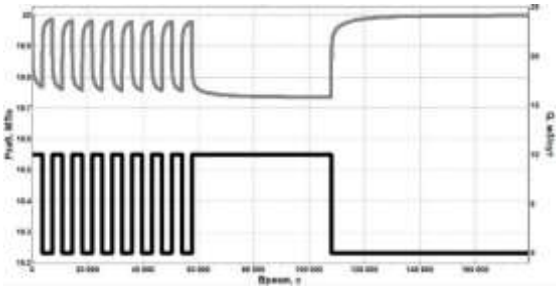
а)



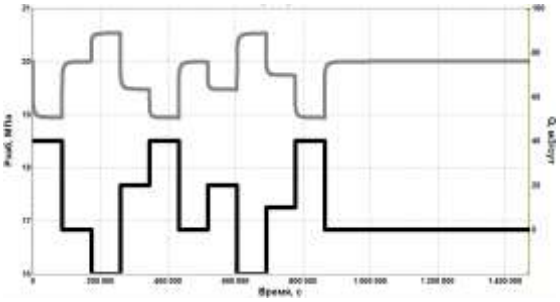
б)



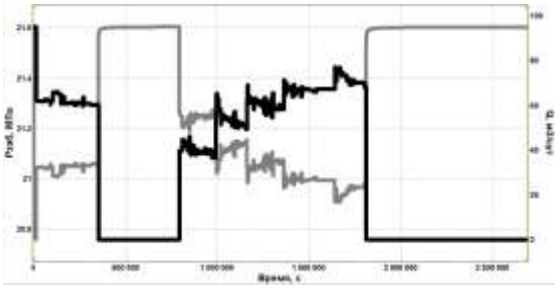
в)



г)



д)



е)

Рис. 7. Примеры различных технологий исследований для проверки эффективности методики;

а) работа скважины с постоянным дебитом, б) две ступени с уменьшением дебита, в) две ступеньки с увеличением дебита, г) периодическая работа скважины, д) скважина с многоступенчатым дебитом, е) скважина с произвольным дебитом

6. Заключение

Создана бета-версия программного продукта для интерпретации гидродинамических исследований скважин с восстановлением предыстории работы скважин. Она включает две интерпретационные модели: однородный бесконечный пласт и радиальный композитный пласт.

Введение в модели нового алгоритма определения предыстории позволяет:

- более точно выбрать интерпретационную модель, лучше оценить фильтрационно-емкостные свойства и геометрию пласта;
- восстановить время и дебит, предшествующий известной истории работы скважины, если он не был достоверным;
- определить эквивалентные время и де-

бит предыстории, создающие такое же возмущение в пласте, как и то, которое обусловлено реальными, но неизвестными дебитами.

Работа выполнена при поддержке гранта РФФИ-18-07-00503А.

Implementation of the Rate History Reconstruction Method for the Non-Stationary Filtering Problem Solving

N.P. Efimova, P.V. Kryganov, I.V. Afanaskin

Abstract. Piezo-conductivity equation analytical solution for a radial composite model is considered, with taking into account the superposition of real and equivalent well rate. Well test interpretation program was developed and tested using history restoring algorithm for two fluid filtration models.

Keywords: welltest, pressure build-up curve, well production history

Литература

1. П.В. Крыганов, Н.П. Ефимова, И.В. Афанаскин, С.Г. Вольпин, Ю.А. Мясников, А.В. Свалов. Повышение достоверности определения фильтрационных параметров пласта при решении задачи нестационарной фильтрации путем учета предыстории работы скважины. «Труды НИИСИ», Т. 9 (2019), № 1, 14-22.

2. Olivier Houze, Didier Viturat, Ole S. Fjaere. Dynamic Data Analysis. V 5.20. Kappa Engineering, 2019.

3. J.S. Olarewaju, W.J. Lee. An Analytical Model for Composite Reservoirs Produced at Either Constant Bottomhole Pressure or Constant Rate. Paper SPE 16763, 1987, 27-30.

4. J. Nocedal, S.J. Wright. Numerical optimization. Springer, New York, 1999.

5. А. Чодри. Гидродинамические исследования нефтяных скважин. М., ООО Премиум Инжиниринг, 2011.

Подписано в печать 5.3.2020 г.
Формат 60x90/8
Печать цифровая. Печатных листов 4
Тираж 100 экз. Заказ №

Отпечатано в ППП «Типография «Наука»
121099, Москва, Шубинский пер., 6