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

ТРУДЫ НИИСИ РАН

ТОМ 11 № 1

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2021

Тематика номера:
Вопросы программирования, моделирование динамических природных процессов, визуализация, информационные и компьютерные технологии

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:

Programming issues, modeling of dynamic natural processes, visualization, information and computer technologies

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

О некоторых особенностях JWT аутентификации в веб-приложениях

А.Б. Бетелин[1], И.Б. Егорычев[2], А.А. Прилипко[3], Г.А. Прилипко[4], С.Г. Романюк[5], Д.В. Самборский[6]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ab@niisi.msk.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, egorychev@niisi.msk.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aaprilipko@niisi.msk.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, prilipko@niisi.msk.ru;

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

[6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, samborsky_d@fastmail.com

Аннотация. Наличие надежного механизма аутентификации является важным требованием при построении веб-приложений, работающих с защищенными данными. Использование токенов для организации доступа позволяет упростить архитектуру сервера, повысить производительность и дает возможность одновременной работы с различными сервисами и доменами. В рамках данной работы рассмотрены некоторые преимущества и особенности использования JWT токенов, а также приведен пример реализации механизма аутентификации на их основе.

Ключевые слова: JWT, аутентификация, безопасность, веб-приложение

1. Введение

В веб-приложениях, как правило, присутствуют элементы, доступ к которым требуется ограничить. Примером таких элементов может быть личный кабинет пользователя или функциональность, доступная только после оплаты. Чтобы получить доступ к подобным ресурсам, пользователю необходимо пройти проверку прав, т. е. процедуру авторизации. При этом приложение должно убедиться в подлинности пользователя, которому предоставляется доступ, т. е. осуществить аутентификацию.

Один из способов аутентификации в веб-приложениях заключается в использовании сессий. В этом случае после успешного входа пользователя в систему сервер генерирует уникальный идентификатор, сохраняет его в базе данных в привязке к пользовательским данным и отсылает его на клиентскую часть приложения. В дальнейшем при обращениях к серверу клиент прикрепляет этот идентификатор к запросу. Получив запрос с указанием сессионного идентификатора, сервер извлекает из базы данных сведения о контексте работы пользователя и обрабатывает поступивший запрос с учетом этой информации. У такого способа есть ряд недостатков. Прежде всего, необходимость осуществления операций с базой данных накладывает дополнительную нагрузку на сервер и усложняет его архитектуру. Кроме того, в случае сервисной распределенной архитектуры системы ведение такой базы данных может ограничивать возможности масштабирования и увеличивать количество сетевых запросов.

Одной из альтернатив использования сессий для реализации аутентификации в веб-приложении является применение токенов доступа. В данной статье рассматривается пример использования JWT (JSON Web Tokens) – токенов доступа на основе формата JSON. Рассмотрим схему работы механизма аутентификации на их основе (см. Рис. 1).

На стороне клиента пользователь вводит данные для входа в систему, которые затем

Рис. 1. Схема клиент-серверного взаимодействия

передаются на сервер. Сервер проверяет полученные данные и в случае успешной аутентификации передает на клиентскую часть специальный уникальный ключ – токен, содержащий информацию о пользователе. Этот токен может быть затем сохранен во временном хранилище в браузере – *local storage* – и использован при последующих обращениях к программному интер-

фейсу сервера (API). При каждом таком обращении в состав запроса включается копия полученного ранее токена. Сервер, получив запрос, дешифрует находящийся в нем токен и по содержащейся в нем информации проверяет, имеет ли пользователь, приславший запрос, доступ к запрашиваемым данным. Если проверка завершается успешно, сервер в ответ высылает запрошенные защищенные данные. Когда пользователь осуществляет выход из системы, токен удаляется из временного хранилища. Кроме того, для токена можно установить период, по истечении которого он считается просроченным и непригодным для осуществления запросов к защищенным данным. Если в составе клиент-серверного запроса токен отсутствует или просрочен, для выполнения API-обращений к серверу пользователю требуется заново осуществить вход в систему, чтобы получить новый токен.

2. Другие часто используемые варианты аутентификации

Существует целый ряд способов и протоколов аутентификации, в том числе и не основанных на использовании токенов доступа [1]. Для первичной аутентификации наиболее простым способом является применение паролей. В этом случае пользователь обычно идентифицирует себя при помощи имени или адреса электронной почты, вводит пароль и в зависимости от настроек сервера либо получает идентификатор открытой пользовательской сессии, либо, в более простом случае, ко всем его последующим запросам присоединяется HTTP заголовок *Authorization*, содержащий данные для входа. Главным недостатком парольной аутентификации считается ее ненадежность, так как есть высокий риск утечки и взлома пароля, поскольку пользователи склонны устанавливать простые и неуникальные пароли, а также не следить за соблюдением их секретности.

Повысить надежность парольной аутентификации позволяет применение механизма 2FA (two-factor authentication) – двухфакторной аутентификации или аутентификации при помощи одноразовых паролей. Этот механизм подразумевает предоставление пользователем дополнительной к паролю информации – одноразового кода, который может быть получен в виде смс на номер мобильного телефона, сгенерирован аппаратным или программным способом либо выпущен заранее на носителе, например, на специальной карточке с кодами. Применение второго фактора позволяет значительно повысить уровень безопасности и может быть востребовано в определенных видах приложений (например, банковских) или при выполнении определенных действий.

Избежать передачи пароля между клиентом и веб-сервером позволяет сетевой протокол аутентификации Kerberos [10], основанный на взаимной аутентификации клиента и сервера при помощи доверенного посредника – специальной службы, которая выпускает сеансовые ключи и мандаты, используемые для установки связи между клиентом и сервером.

Надежным, но и достаточно сложным в реализации и использовании является способ аутентификации по сертификатам. В случае его применения посредником между клиентом и сервером выступает удостоверяющий центр (Certification authority, CA, центр сертификации). Он выпускает сертификат, представляющий собой набор данных для идентификации клиента, и гарантирует его подлинность при помощи цифровой подписи. Факт владения сертификатом подтверждается закрытым ключом, который хранится у владельца сертификата. Механизм аутентификации при помощи сертификата стандарта X.509, традиционно используемого в веб приложениях, является частью протокола SSL/TLS и поддерживается большинством современных браузеров. Во время аутентификации сервер проверяет у сертификата подпись доверенного удостоверяющего центра, срок его действия и наличие факта отзыва по списку исключений. Основные трудности такого рода аутентификации связаны со сложностью распространения и поддержки сертификатов.

Упомянутые выше способы и протоколы аутентификации в основном применяются при взаимодействии клиентской части приложения с веб-сервером. В свою очередь использование JWT токенов позволяет осуществить аутентификацию и авторизацию пользователя в серверной части веб-приложения за счет включения в токен полезной информации, характеризующей клиента.

3. Структура JWT токена

Рассмотрим подробнее структуру JWT токена. Он представляет собой строку, сформированную по специальным правилам из трех частей [2]. В первой части содержится заголовок токена, включающий в себя служебную информацию и являющийся JSON объектом со следующими полями:

«typ» – тип. Это поле необязательное и оно может быть включено в состав заголовка при одновременном использовании токена с другими объектами, имеющими схожий заголовок. Рекомендуется указывать в этом поле значение «JWT»;

«alg» – алгоритм хэширования. Значение

этого поля определяет, какой алгоритм будет применяться для шифрования полезного содержимого токена при формировании цифровой подписи. Пример значения поля – «HS256»;

«cty» – тип содержимого. Этот параметр не рекомендуется использовать в обычных случаях. Он должен быть представлен тогда, когда структура токена нетипична и включает в себя вложенные элементы. В случае использования значение поля должно быть «JWT».

Вторая часть токена содержит передаваемую полезную информацию. Она также формируется на основе JSON объекта и может содержать как стандартные, так и произвольные поля. Обычно в этот объект в качестве произвольной информации включаются идентификатор пользователя и идентификатор его роли. Однако состав произвольных данных может быть иным и зависит от цели, с которой выпускается токен. В число стандартных полей могут входить следующие:

«iss» – идентификатор стороны, выпустившей токен;

«sub» – идентификатор стороны, о которой содержится информация в токене;

«aud» – массив идентификаторов получателей токена;

«exp» – момент времени, после наступления которого токен не должен приниматься и обрабатываться;

«nbf» – момент времени, до наступления которого токен не должен приниматься и обрабатываться;

«iat» – момент времени, когда токен выпущен;

«jti» – уникальный идентификатор токена.

Следует отметить, что в качестве ключей стандартных полей в объектах JSON используются короткие строки. Такое правило позволяет сделать итоговое представление токена более компактным.

Третья часть токена представляет собой его цифровую подпись. Для ее вычисления используется алгоритм Base64URL [3]. Этот алгоритм отличается от Base64 тем, что в нем символ «+» заменяется на «-», символ «/» заменяется на «_», а символ «=» опускается, поскольку количеством символов «=» кодируется длина строки и при обратном преобразовании это значение может быть вычислено. При помощи Base64URL вначале первые две части токена кодируются, а затем полученные строки соединяются через символ «.» и итоговая строка хэшируется с использованием алгоритма, определенного в поле «alg» заголовка токена. Полученный хэш и является цифровой подписью токена.

Каждая из трех частей токена кодируется при помощи алгоритма Base64URL, а полученные строки соединяются через точки. Сформированная в результате такого преобразования строка и является JWT токеном (см. Рис. 2).

```
Заголовок  Данные  Подпись
XXXXXXXXX.YYYYYYYY.ZZZZZZZZ
```

Рис. 2. Структура JWT токена

Токен подобного вида удобен для передачи по HTTP и в HTML, поскольку имеет небольшой размер и содержит только допустимые символы.

4. Преимущества JWT аутентификации

Аутентификация на основе токенов имеет ряд важных преимуществ, особенно в системе с сервисной распределенной архитектурой.

Прежде всего, поскольку в токене может содержаться вся необходимая информация для авторизации пользователя в сервисе, не возникает необходимость в дополнительных обращениях к серверу аутентификации. Токен, в свою очередь содержит цифровую подпись этой информации. Сверяя цифровую подпись токена, сервис надежно определяет подлинность информации о пользователе.

Сервер аутентификации в простом варианте использования может не сохранять информацию о выданных токенах, за счет чего уменьшается размер программного кода и упрощается архитектура базы данных. При аутентификации такого рода работа с базой данных включает в себя преимущественно операции на чтение, что способствует повышению скорости работы и расширяет возможности для масштабирования архитектуры.

Кроме того, токен, выданный доверенным сервером, может быть принят другими сервисами и доменами. Таким образом, пользователь, имеющий подобный токен, получает доступ одновременно к целому ряду ресурсов, распределенных среди различных серверов, принимающих этот токен.

Помимо доступа к защищенным ресурсам JWT может быть использован совместно с протоколом и стандартом аутентификации OpenID Connect [4] для передачи профиля пользователя сервиса при организации доступа к внешним сайтам.

5. Пример JWT токена

Допустим, заголовок токена представляет собой следующий объект:

header = {
"alg":"HS256",
"typ":"JWT"

},

а полезная информация заключена в такой объект:

payload = {
"sub":"123",
"name":"user",
"iat":1595949022
}.

В этом случае цифровая подпись может быть вычислена следующим образом:

signature = HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret).

Если в качестве секретного ключа secret задать строку «testjwt», то в результате JWT токен будет выглядеть так:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMiLCJuYW1lIjoidXNlciIsImlhdCI6MTU5NTk0OTAyMn0.rVQT5jNgEHlEzp0WexGWg8XDtZyMN_bgFafi3mIyaUQ

6. Пример реализации JWT аутентификации

Рассмотрим основные этапы реализации JWT аутентификации на примере комплекса информационных систем, применяемых для ведения учета в НИИСИ РАН. Здесь используется распределенная сервисная архитектура, основными элементами которой являются сервер аутентификации, а также сервисы взаимодействия с АСУ и системой бухгалтерского учета. Применение сервера аутентификации позволяет снизить нагрузку на сервисы и повысить тем самым скорость их работы.

При обращении к сервису пользователь может иметь действующий JWT токен с подлинной цифровой подписью. В этом случае при обработке запроса сервисом обращение к серверу аутентификации не производится. Если пользователь при осуществлении запроса к сервису не предоставил действующий JWT, либо проверка цифровой подписи этого токена не была пройдена, сервис перед обработкой запроса перенаправляет пользователя на сервер аутентификации для осуществления входа в систему и получения действительного JWT токена. Рассмотрим подробнее элементы архитектуры такой системы в части реализации JWT аутентификации на сервере и на клиенте.

6.1 Реализация серверной части

В серверной части приложения должны быть реализованы как минимум три функциональных блока:

блок регистрации пользователя;

блок обработки входа пользователя в систему;

блок обработки запроса к защищенной информации.

Реализация серверной части JWT аутентификации доступна во многих языках программирования и фреймворках. В качестве компонентов сервера аутентификации рассмотрим использование веб-сервера, написанного на языке JavaScript с использованием фреймворка Express [5] и библиотеки jsonwebtoken [6].

При регистрации нового пользователя сервер запоминает его данные и генерирует на их основе токен:

```
const jwt = require('jsonwebtoken')
…
app.post('/register', (req, res) => {
…
const token = jwt.sign({ user }, process.env.SECRET)
res.json({
  token,
  …})
…})
```

Здесь в переменной *user* хранится ссылка на объект, содержащий пользовательские данные, введенные при регистрации, а в переменной окружения *SECRET* сервер хранит секретный ключ и использует его при генерации JWT токенов. Полученный таким образом JWT токен сразу может быть передан на клиентскую часть, если предусмотрена автоматическая аутентификация новых пользователей.

Если пользователь осуществляет вход в приложение, то на стороне сервера выполняется похожий сценарий, с той лишь разницей, что для генерации токена используется считанная из базы данных информация о пользователе, соответствующая данным, введенным при входе в систему

```
app.post('/login', (req, res) => {
…
const token = jwt.sign({ userInfo }, process.env.SECRET)
res.json({
  token,
  …})
…})
```

Здесь в переменной *userInfo* хранится ссылка на объект, содержащий пользовательские данные, найденные в базе данных по ключевой информации, введенной пользователем при входе в систему.

При поступлении запроса к защищенным данным сервиса он считывает из заголовка http-запроса с именем *Authorization* значение JWT токена:

```
app.get('/protected-page', (req, res) => {
…
```

```
const bearerHeader = req.headers['authorization']
const bearerToken = bearerHeader.split(' ')[1]
…
```

Затем производится верификация токена и получение из него полезной информации в расшифрованном виде.

```
jwt.verify(bearerToken, process.env.SECRET,
  (err, decoded) => {
…})
…})
```

Здесь в переменной *decoded* в случае успешной валидации возвращается ссылка на расшифрованную полезную информацию. На основе полученных данных формируется ответ сервера. Если проверка JWT токена и содержащейся в нем полезной информации пройдена успешно, сервис пересылает клиенту запрошенные защищенные данные. В противном случае пользователь перенаправляется на сервер аутентификации для осуществления входа в систему.

6.2 Реализация клиентской части

Рассмотрим, что должно быть реализовано в клиентской части приложения, использующего JWT аутентификацию и веб браузер. После успешного входа в систему клиент получает в ответе от сервера аутентификации JWT токен. Этот токен должен быть сохранен в клиентском окружении для его последующего использования при API запросах к защищенным данным сервисов. Можно использовать различные варианты хранения такого рода пользовательских данных. Например, токен может быть сохранен в данных самого приложения:

```
state.user = userData // {name, token}
```

При таком варианте хранения токен будет доступен в контексте (*state*) текущего состояния приложения до его закрытия. Если требуется сохранение токена в промежутках между запусками приложения, можно использовать хранилище браузера *local storage*:

```
localStorage.setItem('user',
  JSON.stringify(userData))
```

В этом случае при повторном запуске приложения можно осуществить автоматический вход пользователя с использованием данных, прочитанных из хранилища:

```
userData = localStorage.getItem('user')
```

При наличии в данных клиентской части системы JWT токена, полученного либо с сервера, либо из хранилища браузера, во всех запросах к защищенным данным сервера необходимо устанавливать http заголовок *Authorization*. Значением этого заголовка будет являться строка, содержащая сам токен и ключевое слово *Bearer*. Если для осуществления http-запросов используется распространенная библиотека для осуществления асинхронного клиент-серверного взаимодействия axios [7], установить требуемый заголовок можно следующим образом:

```
axios.defaults.headers.common['Authorization'] = `Bearer ${userData.token}`
```

Важно предусмотреть возможность удаления информации о пользователе из локального хранилища браузера. Такая функциональность потребуется для обработки выхода пользователя из системы, а также при получении сообщения от сервера о том, что токен не является действительным.

Очистка пользовательской информации в локальном хранилище браузера может быть произведена следующим образом:

```
localStorage.removeItem('user')
```

7. Возможные уязвимости

При проектировании клиент-серверной системы, применяющей JWT-аутентификацию, важно учесть ряд особенностей, влияющих на безопасность.

Если в заголовке токена в поле *alg* указано значение *none*, то это может означать, что целостность токена уже проверена. Однако возможно, что злоумышленник подменил токен, указав в нем свою полезную информацию и *none* в качестве алгоритма шифрования. Чтобы защититься от такого рода атак, необходимо принимать токены только с явно указанным алгоритмом шифрования.

Кроме того, в поле *alg* может быть неправильно указан тип шифрования [8] ввиду того, что злоумышленник поменял в токене тип шифрования, например, с HMAC на RSA, подписав токен публичным RSA ключом по алгоритму HMAC. В этом случае сервер, руководствуясь содержимым поля *alg* выполнит валидацию токена с использованием публичного ключа RSA, которая произойдет успешно, несмотря на то, что токен был подменен. Поэтому важно при валидации JWT токенов использовать актуальные версии проверенных библиотек, в которых подобная уязвимость отсутствует.

Следует также отметить, что при создании токена необходимо использовать сложный секретный ключ для его защиты от подбора. Поскольку секретный ключ хранится только на сервере, использование в его качестве сложной и достаточно длинной последовательности символов не представляет трудности.

Отдельную проблему может представлять аннулирование (инвалидация) выданных токенов, которая, скорее всего, потребуется в таких ситуациях, как изменение данных пользователя

(в том числе его роли) или же кража токена злоумышленником. Одним из вариантов решения этой проблемы является установка малого значения времени жизни токена в поле *exp*. В этом случае обновление токена будет происходить настолько часто, насколько это необходимо для поддержания актуальности содержащейся в нем информации. Однако частый ввод данных для входа в систему на сервере аутентификации с целью получения нового токена зачастую неудобен. Для устранения этого препятствия можно построить процесс аутентификации таким образом, что пользователь получает от сервера в дополнение к основному токену дополнительный специальный токен, который используется только при получении нового основного токена. Специальный токен имеет длительное время жизни и обновляется каждый раз, когда пользователь запрашивает новый основной токен. Аутентификация пользователя, предоставившего корректный специальный токен, происходит автоматически и незаметно для него. В случае отсутствия специального токена и окончания срока действия основного токена от пользователя потребуется ввод имени и пароля.

Хранение токена в *local storage* браузера потенциально опасно ввиду возможности его утечки при атаке Cross-site scripting (XSS) [9], состоящей во внедрении вредоносного кода в текст страницы. В дальнейшем этот код, имея доступ к клиентским данным, взаимодействует с сервером злоумышленника в целях передачи и неправомерного использования этих данных. В частности, при таком виде атак есть вероятность доступа вредоносного кода и к содержимому *local storage*, в котором могут быть сохранены JWT токены. Следует отметить, что уязвимость перед XSS атакой есть характеристика всего клиентского окружения и не специфична именно для случая реализации JWT аутентификации.

Использование токенов доступа является одним из вариантов реализации технологии единого входа, главный недостаток которой – повышение важности единственного пароля. Для уменьшения вероятности утечки пароля на сервере аутентификации может быть задействован сетевой протокол Kerberos [10]. В случае его использования вся информация, содержащая необходимые для аутентификации сведения и передаваемая между клиентом и сервером, надежно шифруется за счет использования выделенной службы генерации сеансовых ключей KDC (Key Distribution Center). Полученный при аутентификации в системе JWT токен передается в эту службу при обращении для получения сеансового ключа. Служба KDC проверяет подлинность токена и включает его данные в поле *AuthorizationData* ключа. При проверке ключа в сервисе из него извлекается JWT токен и обрабатывается обычным способом.

8. Заключение

Использование JWT токенов для аутентификации в веб-приложениях имеет ряд преимуществ по сравнению с вариантом, предусматривающим хранение на сервере пользовательских сессий, особенно в случае распределенной сервисной архитектуры.

Реализация базовой части такого механизма аутентификации требует доработки как клиентской, так и серверной части приложения, однако не представляет особой сложности, а ее основные элементы приведены в данной статье.

Несмотря на существенные преимущества JWT аутентификации, для обеспечения безопасности системы в целом необходимо учесть ряд особенностей как при реализации клиента в части защиты от XSS атак, так и при создании и обработке токенов на сервере в части выбора надежных механизмов валидации и использования сложных ключей шифрования.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (выполнение фундаментальных научных исследований ГП 14) по теме № 0580-2021-0007 «36.20 Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления.» (Рег.№ 121031300051-3).

Some Features of JWT Authentication in Web Applications

A.B. Betelin, I.B. Egorychev, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. Robust authentication is an essential part of many web applications that process protected data. Use of JWT tokens for access control simplifies backend architecture and allows interoperability between identity and service providers. This paper describes advantages and potential weaknesses of JWT tokens and provides an example of an authentication framework.

Keywords: JWT, authentication, security, web application

Литература

1. Д. Выростков. Обзор способов и протоколов аутентификации в веб-приложениях. https://habr.com/ru/company/dataart/blog/262817/.
2. Стандарт “RFC 7519. JSON Web Token (JWT)”. https://tools.ietf.org/html/rfc7519.
3. Стандарт “RFC 4648. Base 64 Encoding with URL and Filename Safe Alphabet”. https://tools.ietf.org/html/rfc4648#section-5.
4. О.М. Бакунова, А.А. Корзун, Н.С. Колосенко, Т.И. Малиновская. Stateless авторизация с использованием JWT. «Web of Scholar», Т.1 (2018), 6(24).
5. Сайт “Express – Node.js web application framework”. https://expressjs.com.
6. Сайт “JsonWebToken implementation for node.js”. https://github.com/auth0/node-jsonwebtoken.
7. Сайт “Promise based HTTP client for the browser and node.js”. https://github.com/axios/axios.
8. McLean T. Critical vulnerabilities in JSON Web Token libraries. https://www.chosen-plaintext.ca/2015/03/31/jwt-algorithm-confusion.html
9. KirstenS. Cross Site Scripting (XSS) Software Attack. “OWASP Foundation” https://owasp.org/www-community/attacks/xss/
10. Стандарт “RFC 4120. The Kerberos Network Authentication Service”. https://tools.ietf.org/rfc/rfc4120.

Мультиплексирование доступа к USB-устройствам в среде виртуализации GNU Linux/QEMU/KVM/Libvirt

А.Б. Бетелин[1], А.Г. Ванин[2], И.Б. Егорычев[3], А.А. Прилипко[4], Г.А. Прилипко[5], С.Г. Романюк[6], Д.В. Самборский[7]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ab@niisi.msk.ru;

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

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, egorychev@niisi.msk.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aaprilipko@niisi.msk.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, prilipko@niisi.msk.ru;

[6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, sgrom@niisi.ras.ru

[7]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, samborsky_d@fastmail.com

Аннотация. Возможность дистанционного подключения USB-устройств к виртуальным операционным системам является важным требованием при внедрении систем виртуализации. Программное обеспечение среды виртуализации GNU Linux/QEMU/KVM/Libvirt и программы удаленного доступа к экранам виртуальных ОС включают только базовые средства для использования локально- и дистанционно-подключенных USB-устройств. Они имеют лишь ограниченные возможности физического переключения USB-устройств, и не допускают их совместного использования несколькими виртуальными ОС. В рамках данной работы была выполнена доработка программы сервера USB-устройств и предложен управляемый TCP-мультиплексор, который выполняет переключение соединений USB-устройств с виртуальными ОС.

Ключевые слова: USB, виртуализация, QEMU, usbredir, Linux

1. Введение

Интерфейс подключения периферийных устройств Universal Serial Bus (универсальная шина последовательного интерфейса передачи данных, USB) был предложен в 1996 году консорциумом нескольких крупных производителей вычислительной техники и программного обеспечения. Благодаря запасу производительности и относительно простой организации обмена данных интерфейс USB позволил заменить целый набор последовательных и параллельных интерфейсов (RS232, LPT, PS/2, IEEE 1394, MIDI), используемых для подключения периферийных устройств персональных компьютеров. Популярность нового интерфейса объяснялась возможностью «горячего» подключения устройств и простотой устройства кабеля: в нем была всего одна дифференциальная витая пара для передачи данных, линия питания +5V, и нулевой провод. Линия +5V также позволила подавать питание для маломощных устройств непосредственно через USB-кабель.

В течение последующих лет спецификации стандарта USB несколько раз дополнялись, чтобы удовлетворять растущие потребности в скорости передачи данных. Новые версии стандарта обеспечивают физическую и функциональную совместимость при подключении, если USB- разъемы физически совместимы, то контроллер и периферийное устройство обязаны найти максимальную версию протокола, по которому они смогут работать, см. версии стандарта 1.x, 2.0, 3.x [1]. Недавно принятый стандарт USB4 совместим только с версиями USB 2.0 и 3.x, но зато он может работать в режимах интерфейсов DisplayPort, PCI Express, и Thunderbolt 3. Поскольку стандарт USB4 по сути является совершенно новой технологией, мы его здесь не рассматриваем, далее под термином USB будут подразумеваться стандарты USB версий от 1.x до 3.x.

Для изучения стандарта USB могут быть использованы как официальные спецификации от консорциума производителей USB Implementers Forum [1], так и неофициальные справочные пособия, например [2]. Шина USB образует сеть с топологией «активное дерево», где листьями являются оконечные устройства, в промежуточных узлах находятся концентраторы (хабы), а корневой хаб напрямую управляется USB-контроллером, или хостом. USB-хабы могут образовывать до пяти уровней дерева, не считая

корневого. Одновременно на одной USB-шине может работать не более 127 устройств (включая хабы), потому что поле адреса устройства содержит 7 бит. В простейшем случае единственный хаб находится непосредственно в персональном компьютере, а устройства подключаются к его USB-разъемам. Дерево шины USB называется «активным», потому что только контроллер может инициировать обмен данными, а оконечные устройства отвечают на прямые запросы от контроллера (исключением служит добавленный в версии 2.0 режим OTG, On-The-Go, в котором два USB-устройства могут быть соединены друг с другом без концентратора — тогда одно из устройств фактически выполняет роль концентратора). Такой подход исключает конфликты доступа к шине и позволяет построить временной план передачи сообщений: в каждый момент времени заранее известен абонент и направление передачи данных (для полудуплексного режима в версиях 1.x/2.0, а в версиях 3.x добавлены две дифференциальные пары для полнодуплексного режима).

Следует отметить, что USB-хабы тоже выступают в роли оконечных устройств, которым передаются сообщения от контроллера. Именно с помощью таких сообщений контроллер выполняет первичную инициализацию сети USB-шины, а затем узнает о подключении новых устройств. Также топология дерева оказывается удобной в случае, когда одно физическое USB-устройство нужно представить в виде нескольких логических устройств, например, видеокамеру как два отдельных USB-устройства для передачи видео- и аудио-данных. Тогда это устройство называется составным (compound device), оно содержит в себе хаб (который может быть логическим, т.е. созданным программно) и несколько подключенных к нему логических USB-устройств.

Обмен данными по USB-шине выполняется в виде транзакций, где каждая транзакция инициируется хостом в отношении некоторого USB-устройства и состоит из передачи нескольких пакетов в одном или обоих направлениях. Каждый пакет начинается со специальной синхропоследовательности, состоящей из 8 (для режима Low / Full speed) или 32 (для режима High speed) битов, которые опознаются приемником и используются для подстройки фазы тактового генератора приемника. Далее следуют 4 бита поля PID (Packed IDentifier), обозначающего тип пакета. Все следующие данные зависят от типа пакета, последним полем может быть контрольная сумма (CRC5 или CRC16), а заканчивается каждый пакет особым сигналом дифференциальной пары, который отличается от обычной передачи данных и может быть схемотехнически опознан.

Первым в транзакции всегда передается короткий токен-пакет, содержащий адрес USB-устройства и номер его приемной точки (endpoint, см. ниже). PID-поле токена также указывает на вид данной транзакции: ввод данных, вывод данных или служебная транзакция. Следующий пакет обычно имеет тип DATA и содержит данные, после него посылается пакет подтверждения приема данных. Количество данных в DATA-пакете не определено заранее, и может быть от 0 до максимально допустимого размера пакета данных (определен в параметрах endpoint). Если нужно передать много данных, то в течение одной транзакции посылается несколько пакетов DATA так, что размер всех пакетов, кроме последнего, максимален, а неполный пакет обозначает завершение передачи.

Например, если требуется передать порцию данных USB-устройству, то хост сначала посылает токен-пакет с идентификатором PID, равным коду OUT, затем один или несколько DATA-пакетов и далее ожидает ответа от устройства. Пакет подтверждения содержит один из трех кодов: ACK, сообщающий об успешном приеме данных; NACK, сообщающий о поврежденных данных или временной невозможности принять данные; STALL, обозначающий некоторую ошибку, требующую особой обработки со стороны хоста. Если требуется, наоборот, получить порцию данных от USB-устройства, то хост посылает токен-пакет с идентификатором PID, равным коду IN, затем принимает DATA-пакеты, а потом посылает подтверждение приема. Если устройство не готово посылать запрошенные данные, то оно вместо DATA-пакета посылает пакет подтверждения с кодом NACK. Структура транзакции не обязательно включает все вышеперечисленные типы пакетов. Так, некоторые служебные транзакции не содержат пакетов данных, а транзакции в изохронном режиме передачи данных, используемом для передачи мультимедиа-данных, не имеют завершающих пакетов подтверждения.

Как видно из структуры токен-пакета, адресатом транзакции является не просто USB-устройство, а приемная точка (endpoint) этого устройства. Передача данных между хостом и endpoint образует канал (pipe) передачи данных, параметры которого известны обеим сторонам. Определены четыре типа каналов: Control (управляющий канал), Interrupt (канал прерываний), Bulk (канал для объемных пересылок), Isochronous (изохронный канал для передачи мультимедиа-данных). Каналы создаются при настройке подключенного устройства, в процессе которой устройство сообщает о своих функциях и связанных с ними потребностях по

обмену данных. В момент подключения устройства предполагается, что существует нулевой управляющий канал, через который и выполняется настройка.

Все транзакции на USB-шине планируются хостом и выполняются по расписанию, регулярно повторяясь внутри временных кадров (frame) длительностью 1 мсек в режимах Low Speed / Full Speed или микрокадров (microframe) длительностью 125 мксек в режиме High Speed. О начале нового кадра или микрокадра извещает специальный пакет (Start Of Frame, SOF-пакет). При создании или модификации расписания транзакций хост резервирует временные промежутки для каналов типа Interrupt и Isochronous, оставшееся время используется для транзакций по каналам Control и Bulk. Стандарт требует, чтобы хост резервировал для транзакций Interrupt и Isochronous не более 90% кадра и не более 80% микрокадра (при невозможности выполнить это требование ОС должна отказаться подключать устройство на этой шине и, например, рекомендовать пользователю переключить его на менее загруженную USB-шину).

От типа канала зависит устройство его транзакций, их приоритет при планировании и даже частота сигнала при передаче битов данных. Ниже перечислены основные свойства всех четырех типов каналов:

– Control, управляющий канал. По таким каналам передаются управляющие команды, опрашивается состояние устройств, но могут также передаваться произвольные данные, до 64 байтов в транзакции. Каждая транзакция завершается пакетом подтверждения приема.

– Interrupt, канал прерываний. Этот тип канала предназначен для передачи срочных и коротких сообщений, например, нажатия клавиши клавиатуры или элементарного движения компьютерной мыши. Термин «канал прерываний» здесь не совсем корректен, поскольку протокол USB-шины не разрешает устройствам инициировать обмен сообщениями. Для своевременной доставки сообщений по каналам прерываний хост только гарантирует регулярность опроса устройства. В параметрах канала есть поле, задающее период для транзакций обмена через этот канал, измеряемый в кадрах или микрокадрах и принимающий значения от 1 до 255. Например, сообщения о движении компьютерной мыши, подключенной по USB-шине в режиме Low Speed / Full Speed (т.е. для устройства USB 1.1), могут доставляться с минимальной дискретностью 1 мсек, поскольку опрос можно выполнять не чаще, чем один раз в кадр. Для режима High Speed (USB 2.0, 3.x) минимальная дискретность будет равна длительности одного микрокадра - 125 мксек. Максимальный размер данных в одной транзакции канала прерывания равен 8, 64 или 1024 байтов для режимов Low, Full и High Speed соответственно. Если на момент опроса канала у USB-устройства нет события для хоста, то он вместо пакета с данными отвечает пакетом подтверждения с кодом NACK.

– Bulk, канал объемных пересылок. Через каналы этого типа выполняется однонаправленная посылка или прием данных с проверкой контрольных сумм пакетов, подтверждением доставки и повторной пересылкой в случае сбоев. Ограничения на размер данных в рамках транзакции нет, кроме длительности кадра циклограммы USB-шины. Передачи данных через Bulk-каналы наименее приоритетны, они выполняются в течение времени кадра, оставшегося после временных окон, зарезервированных для Interrupt и Isochronous транзакций, а также после запланированных Control-транзакций. Таким образом, Bulk-каналы не гарантируют ни минимальной пропускной способности, ни максимальной задержки, но обеспечивают целостность переданных данных.

– Isochronous, изохронный канал для передачи мультимедиа-данных. Этот канал применяется для посылки данных с гарантированной пропускной способностью и минимальной задержкой. Данные сопровождаются контрольными суммами, но пакеты подтверждений приема в таких транзакциях не предусмотрены. Подразумевается, что поврежденные данные могут быть отброшены. Хост резервирует временные окна в USB-кадрах для Isochronous каналов данных, чтобы обеспечить потребности USB-устройства по своевременной пересылке данных. Такие каналы также могут использовать нестандартную частоту передачи импульсов на дифференциальной линии. Предусмотрено три типа синхронизации: *асинхронный*, когда частота передачи хоста и устройства не связаны; *синхронный*, когда частота устройства должна совпадать с частотой передачи SOF-пакета; *адаптивный*, когда частота устройства подстраивается под частоту передачи, которую использует хост в стартовом пакете транзакции. Такая возможность была добавлена в основном для передачи оцифрованных аудиосигналов, где устройства могут использовать разную частоту дискретизации в ADC- и DAC- преобразователях. Подстройка скорости передачи данных под скорость оцифровки упрощает обработку цифровых сигналов, уменьшает задержку и сокращает необходимые размеры входных и выходных буферов.

При подключении новых устройств к шине они опознаются хабом и переводятся в состояние сброса (reset). В этом состоянии устройства находятся до тех пор, пока контроллер шины не

приступит их распознаванию. О появлении новых устройств хост узнает от хаба, выполняя регулярный опрос по специальному каналу Interrupt. Распознавание и настройка новых устройств выполняется по одному, даже если было подключено несколько новых устройств одновременно. После того, как хост разрешает хабу снять сигнал сброса, устройство имеет временный нулевой адрес на шине и через него получает транзакции первичной настройки, используя нулевой канал Control. Хост определяет параметры устройства, присваивает ему уникальный номер на шине и создает все необходимые каналы обмена. Поскольку хабы являются такими же USB-устройствами, наравне с оконечными устройствами, их подключение выполняется тем же способом. Следовательно, при подключении нового поддерева хабов и других устройств процедура опознавания и настройки выполняется рекурсивно.

Приемные точки устройства, endpoints, входят в состав интерфейса (interface), который описывает определенную функцию устройства. Описание интерфейса содержит его название, а также коды класса, подкласса и протокола, соответствующие функции устройства, представляемой этим интерфейсом. Интерфейсы служат скорее логическим понятием, они упорядочивают написание драйверов USB-устройств. Кроме того, USB-устройство при инициализации может сообщить описание нескольких доступных режимов их работы, или конфигураций, которые могут отличаться составом интерфейсов и другими атрибутами, например, использованием автономного источника питания или питанием от USB-шины. Большинство устройств имеют только одну доступную конфигурацию, в которую их переводит контроллер в процессе инициализации.

Подводя итог описания программного интерфейса USB-устройства, можно сказать, что в каждый момент он состоит из текущей конфигурации, которая содержит один или несколько интерфейсов, включающих в себя одну или несколько приемных точек. Ядро операционной системы обычно позволяет узнать текущую настройку подключенных USB-устройств, например, ядро ОС Linux показывает USB-устройства, интерфейсы и приемные точки в виде файлов в каталоге /sys/bus/usb/devices.

Среди недостатков стандарта USB обычно упоминают следующие: ограничение длины кабеля (3 или 5 метров в зависимости от версии стандарта), сложность реализации минимально необходимой для удовлетворения стандарта логики функционирования USB-устройств и USB-хабов, дороговизну получения идентификатора Vendor ID и процесса сертификации USB-устройств, отсутствие бесплатного диапазона Vendor ID / Product ID для радиолюбителей и некоммерческих организаций.

Пользователи также иногда жалуются на медленную инициализацию USB-устройств при их подключении, которая может занимать несколько секунд. Но эта задержка не связана с самим протоколом USB-шины, а вызвана таймаутами в драйверах USB устройств разного типа. Эти таймауты обычно добавлялись вынужденно, чтобы работали USB-устройства, которые нарушают протокол и оказываются не готовы к работе сразу после подключения к хабу и снятия сигнала сброса (reset). Например, в феврале 2010 года в ядре ОС Linux таймаут для USB-устройств внешних носителей (умолчательное значение для параметра delay_use драйвера usb_storage) был уменьшен с 5 до 1 секунды в ответ на обновление парка устройств внешних USB-дисков и флэш-дисков. Данную ситуацию поясняет шуточная расшифровка термина «USB-устройство»: «Usually Slightly Broken device» («как правило немного сломанное устройство»).

Есть еще одна проблема, вызванная недоработками утилит для работы с USB-шинами в операционных системах: подключенное USB-устройство может оказаться в состоянии, в котором оно не инициализировано, но его повторная инициализация средствами ОС не выполняется. Обычно в подобное состояние устройство попадает после того, как средствами ОС выполнена команда его «безопасного отсоединения» («eject»). Согласно логике стандарта USB после этого устройство может быть только отключено, и, если это необходимо, то подключено снова. В похожее состояние может попасть USB-устройство после временного сбоя его питания: поскольку не было разрыва линий, USB-хаб не детектирует событий отключения и подключения устройства. К сожалению, USB-драйверы во многих ОС не имеют функции переинициализации только одного подключенного устройства, возможен лишь принудительный сброс состояния хаба с последующей перенумерацией всех подключенных к нему устройств. При обычном использовании USB-устройства его можно просто отсоединить и подключить повторно, но если компьютер, к которому оно подключается, управляется дистанционно и физический доступ к нему затруднен, то такое решение невозможно.

2. Технологии дистанционного использования USB-устройств

Периферийные устройства, имеющие адаптер локальной сети (Ethernet или WiFi), обладают рядом преимуществ по сравнению с USB-

устройствами:

– они могут совместно использоваться многими клиентами;

– доступ к ним может быть расширен за пределы локальной сети с помощью виртуальных приватных сетей (Virtual Private Network, VPN).

Например, офисные принтеры и сканеры часто имеют сетевой интерфейс, но они более дороги, чем их аналоги с USB-интерфейсом, а также требуют дополнительной настройки, поскольку выступают серверами предоставляемой услуги (печати и сканирования, соответственно).

Логическую сеть USB-шины нельзя расширить с помощью локальной сети и получить доступ к USB-устройству другого компьютера. Несмотря на теоретическую осуществимость этого подхода, не было предложено общепринятых способов ни для инкапсуляции USB-протокола отдельных устройств, ни для построения виртуальной USB-шины, которая могла бы быть частично программной, частично аппаратной аналогично виртуальным сетям передачи данных. Возможно, это объясняется тем, что каналы Isochronous и Interrupt требуют гарантированной минимальной пропускной способности и ограничения максимальной задержки передачи пакетов данных, которые среда передачи локальной сети не всегда может обеспечить. Тем не менее, многие USB-устройства ограничиваются работой только в режимах control и bulk, либо их запросы к скорости передачи данных и их задержки не превышают средней скорости и задержки в локальной сети (Ethernet 100Mbit / 1Gbit / 10Gbit).

Одной из попыток сделать «прозрачное» делегирование USB-устройства на уровне базовых драйверов USB-протокола был проект USB/IP [3]. Программные модули USB/IP добавляются к ядру операционной системы — имеется в виду ОС Linux, но есть также экспериментальная версия USB/IP для ОС семейства Windows — и позволяют передавать доступ к выбранному USB-устройству от одного компьютера другому. При этом вмешательство в работу драйверов шин USB производится как на сервере, куда подключено USB-устройство, так и в клиентской ОС. При использовании виртуальных ОС для подключения дистанционного устройства с помощью USB/IP сначала необходимо выбрать, куда устройство будет отображено: либо в виртуальную ОС, но тогда могут возникнуть сложности, если это ОС семейства Windows, либо в ОС хоста виртуализации, а дальше это устройство передается виртуальной ОС средствами гипервизора. В любом случае решение оказывается громоздким и сложным в поддержке. Кроме того, развитие проекта USB/IP остановилось несколько лет назад и поэтому вызывает сомнение его совместимость с будущими версиями ядра Linux.

За неимением стандартного решения в современных операционных системах задача дистанционной работы с USB-устройствами обычно решается одним из следующих способов:

– При использовании *удаленного рабочего стола* иногда можно тому же компьютеру передать доступ к выбранному локальному USB-устройству. Например, такую функцию имеет протокол Miscrosoft RDP: в его базовой версии поддерживаются не все типы USB-устройств, однако расширение RemoteFX USB Redirection позволяет сделать полноценную инкапсуляцию USB-протокола. Аналогичную функцию обеспечивает протокол удаленного стола NoMachine, который работает не только в системах семейства Windows, но и с ОС Linux и Mac OS X. Данный способ предполагает, что устройство подключено к компьютеру, на котором отображается экран рабочего стола. Следовательно, он не может помочь, когда необходимо получить доступ к USB-устройству, подключенному к произвольному компьютеру локальной сети.

– *В средах виртуализации* наряду с созданием виртуальных устройств широко используется техника передачи доступа к реальному устройству виртуальной ОС. Так, соединенное с хост-системой виртуализации USB-устройство может быть передано в монопольное управление виртуальной ОС. В этом случае местонахождение USB-устройства тоже не может быть произвольным, но оно уже не привязано к компьютеру пользователя. Виртуальные ОС в системе QEMU/KVM, в отличие от других сред виртуализации, позволяют также использовать протокол usbredir из проекта Simple Protocol for Independent Computing Environments (SPICE) [4] для подключения удаленных USB-устройств по локальной сети. Это решение было взято за основу в данной работе и будет рассмотрено подробнее ниже.

– *Коммерческие программы* для делегирования доступа к USB-устройствам на уровне операционной системы. Эти программы, аналогично USB/IP, добавляют свои драйверы в ядро операционной системы, но, в отличие от USB/IP, не всегда гарантируют полную поддержку USB-протокола. Кроме того, закрытость кода таких программ не позволяет осуществить аудит безопасности, что ограничивает область их применения.

- *Программно-аппаратные USB-концентраторы*, подключенные к локальной сети и выступающие в роли «серверов USB-устройств». Такие устройства поставляются с набором драйверов для большинства популяр-

ных ОС, а также с программами, обеспечивающими переключение USB-устройств, тем самым обеспечивая их совместное использование в локальной сети. Несмотря на простоту и удобство, этот способ имеет те же недостатки, что и предыдущий: неполная поддержка USB-протокола и невозможность аудита безопасности.

3. Расширение функций дистанционного подключения USB-устройств

Существующая реализация протокола usbredir из проекта SPICE позволяет подключать к виртуальным машинам QEMU любые USB-устройства, подсоединенные к компьютеру, на котором запущена программа usbredirserver. Эта программа использует стандартную библиотеку libusb для доступа к USB-устройствам в системе ОС Linux. Пользователем дистанционного устройства выступает программа виртуализации QEMU, которая использует библиотеку usbredirclient и перенаправляет пакеты USB-протокола на уровень драйверов виртуальных устройств так, как будто они были получены от локально подключенного к исполняемой операционной системе USB-устройства. Таким образом, при использовании протокола usbredir все программы работают в пространстве пользовательских программ (userspace) в отличие от проекта USB/IP, который использует модули ядра. Это делает данную технологию более безопасной и независимой от версий ядра, применяемой на хосте, осуществляющем виртуализацию.

Протокол usbredir полностью инкапсулирует USB-протокол, поэтому обеспечивает дистанционное подключение произвольных USB-устройств. Но алгоритмы работы программ сервера и клиента usbredir таковы, что подразумевается совершенно определенная последовательность действий:

1. USB-устройство подключается к компьютеру.
2. На этом компьютере запускается программа usbredirserver с указанием VID/PID устройства и порта, куда должен быть перенаправлен USB-протокол.
3. Виртуальная ОС QEMU получает USB Redirection TCP-канал с указанием IP-адреса и порта, где запущена программа usbredirserver.

При нарушении этого порядка действий устройство оказывается недоступно в виртуальной ОС, причем никаких диагностических сообщений пользователь не получает. Фактически, нарушается предположение пользователя о том, что, когда включены все компоненты, процесс делегирования устройства должен работать. Необходимо отметить, что этот принцип нарушается и в обычной работе с локально подключенным USB-устройством: после того, как пользователь выполнил команду «безопасного отключения» устройства («eject» или «safe removal»), оно не может быть повторно инициализировано до тех пор, пока не будет физически отключено и повторно подсоединено.

Третий шаг процедуры настройки не обязательно выполнять после шагов 1 и 2, поскольку при использовании параметра «reconnect timeout» виртуальная ОС периодически выполняет попытки подключения к серверу usbredir. Однако, если несколько виртуальных ОС должны иметь возможность подключения к одному USB-устройству, то приходится либо вручную «на лету» перенастраивать доступ работающих ОС, либо требовать, чтобы не более чем одна их этих ОС работала одновременно.

Для пользователей максимально удобна была бы такая настройка программ, чтобы:

– можно было настроить простые правила совместного использования USB-устройств разными виртуальными машинами;

– состояние дистанционных подключений были бы прозрачны и объяснимы;

– не требовалась бы помощь системных администраторов для решения рутинных задач.

4. Доработка сервера USB-устройств

Программа usbredirserver предоставляет доступ к заданному пользователем USB-устройству через определенный TCP-порт. Разрешается только монопольное использование устройства одним клиентом: после отключения соединения с текущим клиентом выполняется подключение другого клиента, который к этому моменту уже может находиться в очереди ожидания подключения к TCP-порту. Однако официальная версия программы usbredirserver не имеет состояния ожидания подключения USB-устройства, поэтому для правильного его включения нужно гарантировать, что USB-устройство подключено заранее, а после его случайного отключения сервер должен быть перезапущен.

Доработка программы usbredirserver, выполненная в рамках данной работы, добавила режим ожидания устройства (дополнительный параметр командной строки, --wait или -w). Это позволило настраивать экспорт устройств в декларативной форме, т.е. установить соответствие между идентификаторами USB-устройств VID:PID и TCP-портами и затем запустить соответствующее число экземпляров процессов

usbredirserver. Гарантировать отказоустойчивость этих процессов, а также выполнять их запуск можно с помощью специально созданного systemd-сервиса.

5. Разработка TCP-мультиплексора для управления доступом к USB-устройствам

Задача дистанционного использования USB-устройства несколькими виртуальными ОС не может быть решена только с помощью сервера USB-устройств, потому что, с одной стороны, программа usbredirserver не имеет функций переключения доступа к USB-устройству между разными клиентами, а с другой стороны, виртуальные ОС QEMU могут быть подключены или отключены от устройства только администраторами среды виртуализации. Пользователи виртуальных ОС могут лишь использовать устройство, пока оно подключено, а также инициировать операцию «eject», которая, тем не менее, не приводит к отключению от сервера и поэтому не может быть использована для передачи устройства другой виртуальной ОС.

Для решения этих задач была разработана программа TCP-мультиплексора, которая является посредником между клиентами и серверами и может переключать TCP-соединения протокола usbredir согласно заданным правилам.

Оказывается, что для этого не нужно вмешиваться в протокол usbredir — программа мультиплексора может работать в режиме посредника (proxy) и разрывать существующее соединение, когда это необходимо, чтобы уступить доступ к серверу USB-устройства другому клиенту. Клиент, соединение которого было оборвано, теряет доступ к USB-устройству, но параметр reconnect timeout заставляет его выполнять повторные попытки подключения, которые отвергаются программой мультиплексора до тех пор, пока устройство не освобождается.

Программа мультиплексора также имеет возможность читать пакеты протокола usbredir и извлекать некоторую информацию о том, как ведется работа с USB-устройством. Например, можно обнаруживать вышеупомянутое «тупиковое» состояние, в которое USB-устройство переводится после команды «eject», и инициировать в этом случае разрыв соединения. Такая настройка не только исключит ситуации, в которых пользователи случайно инициировали отключение устройства (соединение будет восстановлено автоматически после таймаута и работа с устройством снова будет инициализирована), но и позволит клиентам добровольно уступать доступ к устройству, поскольку оно передается следующему клиенту автоматически. Эта функция незаменима, если известно, что переключения с помощью принудительного разрыва соединения недопустимы при работе с USB-устройством данного типа. Кроме того, из данных USB-пакетов можно извлечь информацию об идентификаторе подключенного устройства и его текущей настройке (например, о количестве и типах настроенных endpoints), а также об общем количестве переданных данных в рамках текущего сеанса работы. Эти данные обычно полезны для диагностики неполадок и протоколирования доступа к USB-устройствам.

Программа TCP-мультиплексора получает настройки либо из конфигурационного файла, либо от программы web-сервера, выполняющего роль панели управления дистанционно-подключаемыми USB-устройствами. Эти настройки состоят из описаний доступных серверов USB-устройств и их параметров, а также из списка клиентов и их приоритетов, которые используются для управления доступом.

Описание USB-устройства содержит следующие атрибуты:

– IP-адрес и порт, который TCP-мультиплексор использует для входящих TCP-соединений;

– IP-адрес и порт сервера usbredir, куда перенаправляется соединение;

– атрибут interruptable, определяющий, разрешено ли принудительно разрывать установленное соединение, когда поступил запрос на соединение от более приоритетного клиента (допустимые значения атрибута: True, False или "timeout=<значение_таймаута>", по умолчанию "timeout=30", т.е. 30сек);

– атрибут disconnect_on_eject, определяющий, нужно ли разрывать соединение при отключении устройства средствами виртуальной ОС клиента (допустимые значения атрибута: True или False, по умолчанию True);

– атрибут eject_detection_method, сообщающий метод определения состояния отключенного устройства (допустимые значения атрибута: "timeout=<значение_таймаута>" или "configuration=<номер_конфигурации>", по умолчанию "configuration=0").

Приоритеты клиентов хранятся в отдельной таблице, где каждый клиент может быть определен по IP-адресу, имени виртуальной ОС, либо уникальному идентификатору виртуальной ОС (UUID).

Алгоритм работы состоит в том, что для каждого дистанционного USB-устройства, доступ к которому контролирует TCP-мультиплексор, создается TCP-сокет и затем ожидаются входящие соединения для этого сокета. Если USB-

устройство в данный момент не используется, то запрос на соединение от клиента удовлетворяется и включается режим TCP-proxy — далее все пакеты протокола usbredir передаются от клиента к серверу и обратно. Если же USB-устройство уже используется, то оно все же может быть передано новому клиенту, если этот клиент имеет приоритет строго выше, чем приоритет текущего клиента, и устройство разрешено переключать принудительно, что определяется атрибутом interruptable. В этом случае существующее TCP-соединение обрывается и устанавливается новое соединение. Значение атрибута interruptable, задающее таймаут (например, значение по умолчанию значение timeout=30), ставит условием переключения период неактивности соединения, в течение которого средняя скорость передачи данных была не выше, чем 1.5x от минимальной средней скорости этого соединения. Такая настройка позволяет в большинстве случаев избегать разрыва соединения во время активной работы с устройством.

Программа TCP-мультиплексора написана на языке Python с использованием библиотек socket, threading и queue. Для управления доступом к каждому USB-устройству запускается специальный поток управления. Он, в свою очередь, запускает отдельный поток управления для обслуживания соединения с новым клиентом. Если нужно уступить доступ к USB-устройству другому клиенту, то этот поток управления получает команду о необходимости закрытия соединения через специальную очередь сообщений.

Кроме того, поток управления, выполняющий функции TCP-proxy, сканирует заголовки всех пакетов протокола usbredir, чтобы обнаруживать переход USB-устройства в состояние «ejected». Стандарт USB не определяет того, в какое именно состояние должно переводиться USB-устройство, когда пользователь операционной системы инициировал его отключение. За это отвечают либо стандартные драйверы операционной системы для устройств данного типа, либо драйвер конкретного USB-устройства. Обычно команда «eject» вызывает следующие действия:

– завершаются открытые транзакции передачи данных (например, сбрасывается буфер записи устройства хранения);

– закрываются все интерфейсы, отменяются все ранее определенные конфигурации устройства и включается специальная конфигурация #0, которая означает, что устройство не инициализировано;

– прекращается обмен любыми USB-пакетами с устройством.

Видно, что после этих действий работа с устройством может быть продолжена только после его физического отключения или после пересканирования всей USB-шины, так как в соответствии со стандартом устройство не может инициировать передачу, а операционная система считает, что работа с этим устройством завершена. Тупиковость этого состояния была осознана разработчиками сервера usbredir, которые предусмотрели сохранение устройства в рабочей конфигурации даже когда клиент вызвал команду «eject». Это все же не позволяет текущему клиенту продолжить работу с устройством, поэтому здесь помогает функция disconnect_on_eject TCP-мультиплексора, заставляющая обрывать соединение и тем самым инициировать повторное подключение устройства.

Способ определения состояния отключенного устройства (атрибут eject_detection_method) может либо указывать на конфигурацию, включение которой означает, что устройство было отключено (например, configuration=0), либо задавать таймаут для соединения — например, timeout=10, тогда устройство будет считаться отключенным после 10 секунд отсутствия обменов USB-пакетами.

Определение приоритета клиента затрудняется тем, что протокол usbredir не содержит никаких идентификаторов клиента (имени или UUID виртуальной ОС), так что программе TCP-мультиплексора доступны только параметры сокета, IP-адрес и порт сокета, который создал процесс QEMU для соединения с устройством. В этом случае используется один и тот же IP-адрес для всех виртуальных ОС, запущенных на одном хосте виртуализации. Тогда для более точного назначения приоритета клиента нужно знать имя виртуальной ОС, установившей соединение. Это можно сделать с помощью системных утилит netstat, ssh, ps и virsh, но только при наличии достаточных прав на хосте виртуализации. Если программа TCP-мультиплексора не имеет этих прав, то для клиента используется тот приоритет, который соответствует его IP-адресу.

Необходимо особо отметить, что алгоритм программы TCP-мультиплексора опирается только на пассивный просмотр данных протокола usbredir и на возможность безопасного отключения с помощью обрыва TCP-сокета, после которого ни клиент, ни сервер не попадают в некорректное состояние. При этом предполагается, что клиенты периодически повторяют попытки подключиться и могут восстановить соединение, когда устройство становится снова доступным. Если у клиента не включена опция reconnect timeout для подключенного USB-устройства, то после первого разрыва соединения оно уже не восстановится до полного выключения этой виртуальной ОС. Хотя протокол

usbredir не имеет специальной функции для «приостановки» работы соединения, можно было бы вмешаться в обмен пакетами и послать клиенту сообщения о неожиданном отсоединении USB-устройства, а серверу — о завершении работы с устройством по инициативе клиента. Позже, когда нужно продолжить работу соединения, достаточно послать пакеты, соответствующие подсоединению временно отключенного устройства. Такой способ был проверен, но не был использован в программе TCP-мультиплексора, поскольку вмешательство в протокол потенциально опасно и может нарушить работу как клиента, так и сервера. Например, при неверной последовательности пакетов протокола usbredir наблюдались случаи аварийного завершения процесса всей виртуальной ОС QEMU, что, наиболее вероятно, является ошибкой библиотеки клиента usbredir.

6. Панель управления для системы совместного использования USB-устройств

Для завершения поставленных выше задач по упрощению использования дистанционных подключений к USB-устройствам дополнительно была создана программа панели управления мультиплексированием USB-устройств. Это отдельная программа на языке Python, выступающая в роли автономного web-сервера, и предоставляющая web-интерфейс для отображения состояния TCP-мультиплексора и управления им.

Данная панель служит для настройки параметров доступа к устройствам и отображения текущего состояния установленных соединений. Пользователь панели может обладать правами администратора, тогда ему доступны все настройки, либо представлять клиента, тогда он может только просматривать все настройки и наблюдать за состояниями соединений, а менять ему разрешено исключительно параметры соединений его виртуальной ОС. Пользователю, обратившемуся из виртуальной ОС, фактически разрешено только временно понижать свой приоритет и разрывать соединение с устройством. Но этих действий оказывается достаточно для того, чтобы уступить некоторое устройство по договоренности с другим пользователем.

Кроме этого, отображение параметров текущих соединений, таких как имя, идентификатор и тип USB-устройства, имя использующей его виртуальной ОС и интенсивность обмена данными, дают общее представление о состоянии совместно используемых USB-устройств и позволяют пользователям планировать и выполнять их переключение без помощи системных администраторов.

7. Заключение

Дистанционное и совместное использование устройств не предполагалось стандартом USB и поэтому такие функции не были реализованы на уровне общесистемного программного обеспечения. В данной работе продемонстрировано, что путем относительно простых доработок программного комплекса можно существенно упростить работу с USB-устройствами в среде виртуализации GNU Linux/QEMU/KVM/Libvirt.

Предложенный программный комплекс не имеет средств авторизации и защиты данных. Предполагается, что в служебной сети, где работают хосты виртуализации и где включены USB-устройства, нет неавторизированных пользователей, а неавторизованный доступ из виртуальных ОС невозможен, так как для их сетевых интерфейсов запрещена маршрутизация пакетов в служебную сеть. На практике эти требования трудно соблюсти, поэтому нужны уровни авторизации доступа к USB-устройствам и защиты соответствующего им трафика от сканирования.

Сервер USB-устройств (программа usbredirserver) не поддерживает TLS уровня авторизации и защиты TCP-соединений — возможно потому, что чаще всего протокол usbredir используется в составе программы удаленного доступа SPICE и инкапсулируется в защищенном соединении вместе с другими потоками данных. С другой стороны, виртуальные ОС QEMU имеют поддержку TLS для chardev каналов, через которые подключаются USB-устройства. Поэтому обеспечить контроль доступа и защиту передаваемой информации могло бы дополнение программы TCP-мультиплексора поддержкой входящих соединений, защищенных TLS-сертификатами.

Исходящие же соединения от TCP-мультиплексора могут быть также защищены другими средствами, например, с помощью защищенной сети VPN или туннеля, установленного через SSH-соединение. Такой уровень защиты потребует только дополнительных действий по начальной настройке и не усложнит интерфейс пользователя программного комплекса.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (выполнение фундаментальных научных исследований ГП 14) по теме № 0580-2021-0007 "36.20 Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления." (Рег.№121031300051-3).

Mutiplexing of USB Devices in GNU Linux/QEMU/KVM/Libvirt Environment

A.B. Betelin, A.G. Wanin, I.B. Egorychev, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. Remote access to USB devices from virtual operating systems could be essential for deployment of virtualization environments. The core components of the GNU Linux/QEMU/KVM/Libvirt virtualization environment and common remote desktop clients provide only basic functions for connection of local and remote USB devices. They offer limited options for physical reconnection of USB devices and no possibility of sharing them. In this work we improve USB device server program and propose a controllable TCP multiplexer that switches connections of USB devices to virtual operating systems.

Keywords: USB, virtualization, QEMU, usbredir, Linux

Литература

1. Сайт "USB Implementers Forum, Inc.", https://www.usb.org.
2. Сайт книги "Peacock, C. (2005) USB in a Nutshell, Making Sense of the USB Standard", https://www.beyondlogic.org/usbnutshell/usb1.shtml.
3. Hirofuchi, Takahiro, Eiji Kawai, Kazutoshi Fujikawa, and Hideki Sunahara, "USB/IP — A peripheral bus extension for device sharing over IP network." In Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 42-42. USENIX Association, 2005.
4. Сайт "SPICE", https://www.spice-space.org/features.html.

Особенности проведения гидродинамических исследований на группе нефтегазоконденсатных месторождений Краснодарского края

А.Г. Дяченко[1], Ю.М. Штейнберг[2], М.Ю. Ахапкин[3]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, adyachenko@bk.ru;

[2] ФГУ ФНЦ НИИСИ РАН, Москва, Россия, yurash@gmail.com

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

Аннотация. Статья описывает результаты гидродинамических исследований, выполнявшихся в скважинах, вскрывших газоконденсатную залежь. Рассматриваются особенности полученных результатов. Приводится информация по методам интерпретации таких исследований и применимость для условий исследуемого объекта.

Ключевые слова: гидродинамические исследования, газоконденсатная залежь

1. Введение

Ряд нефтегазоконденсатных месторождений Краснодарского Края имеют сходное геологическое строение и одинаковые технологические способы их эксплуатации. Гидродинамические исследования скважин и пластов на этих месторождениях позволяют определить фильтрационно-емкостные свойства (ФЕС) эксплуатируемых объектов. Эти данные могут использоваться для расчетных гидродинамических моделей промышленных объектов месторождения. По большинству эксплуатационных скважин проводятся исследования методом «установившихся отборов» и по их данным строятся индикаторные диаграммы (ИД). По характеру поведения этих ИД можно судить об изменениях коэффициента продуктивности в течение времени при эксплуатации продуктивной залежи. По степени наклона кривой ИД отмечают темпы снижения коэффициента продуктивности при длительной эксплуатации скважин.

2. Исходные данные, методы исследований и интерпретации

2.1. Исследования методом установившихся отборов

Исследования, результаты которых приведены далее, проводились преимущественно посредством метода установившихся отборов. На основе этих данных строились индикаторные диаграммы, некоторые из которых приведены на рис. 1.

При рассмотрении этих ИД, построенных в координатах забойное давление ($P_{заб}$) – дебит (q), видно, что большинство из них строились всего по нескольким начальным точкам, и они имеют вид прямых линий. Такая технология исследований при дальнейшей работе скважины на режимах с последовательным увеличением дебита ограничивает получение информации о дальнейшем поведении индикаторных диаграмм в исследуемых интервалах (Рзаб – q), в том числе это не позволяет определить давление, при котором начинается падение коэффициента продуктивности.

С течением времени эксплуатации скважины пластовое давление снижается, а также меняется вид индикаторных диаграмм. В качестве примера на рис.2 приведены ИД, зарегистрированные в скважине № 2п в нефтяной залежи, пачки VII0 в течение 2012, 2015 годов. ИД зарегистрированные до 2015 года имеют вид прямых линий, однако начиная с 2015 года, у ИД наблюдаются отклонения от прямой линии.

На индикаторных диаграммах, зарегистрированных в августе и сентябре 2015 года отчетливо наблюдаются ярко выраженные изгибы линий ИД к оси дебитов. Такое поведение ИД говорит о падении коэффициентов продуктивности по мере увеличения депрессий, которое вызвано значительным снижением забойного давления ниже давления насыщения нефти газом ($P_{нас}$). В результате в призабойной зоне скважины происходит выделение газа, что приводит к снижению проницаемости коллектора призабойной зоны пласта по нефти и увеличению скин-фактора.

При исследовании газоконденсатной залежи построение индикаторных диаграмм производится в координатах $(P^2_{пл}-P^2_{заб})/q$ от q. Большинство индикаторных диаграмм в данных координатах имеют вид прямых линий. По полученным

дебитам жидкости и забойным давлениям графическим методом определяются коэффициенты А и В в уравнении притока к забою скважины. С течением времени эксплуатации газоконденсатной залежи меняется вид индикаторных диаграмм. Например, на рис. 3 представлены ИД, зарегистрированные в скважине № 1п в газоконденсатной залежи, в течение 2001, 2005, 2011 и 2013 годов.

От всех остальных ИД резко отличается индикаторная диаграмма, построенная по исследованию, проведенному в этой скважине в августе 2013 г. Как, видно по этой ИД, с увеличением дебита газа ордината графика падает, что не должно быть при нормальной работе скважины. Данное явление вызвано тем, что забойное давление в скважине во время работы опустилось ниже давления начала конденсации. В этом случае выпадение жидкого конденсата в призабойной зоне скважины приводит к снижению фильтрационных параметров призабойной зоны пласта в скважине для газа.

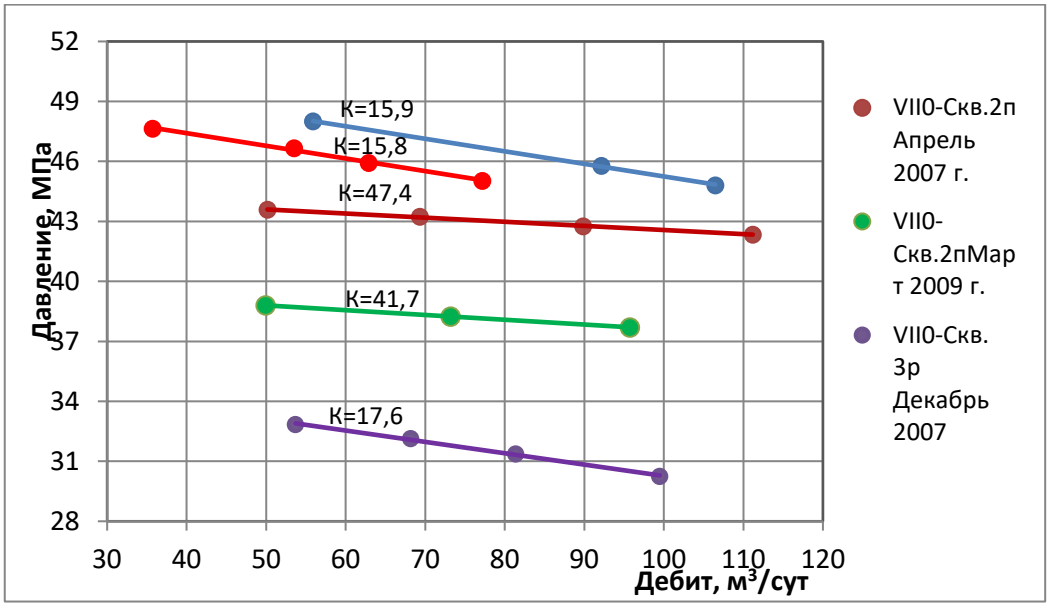

Рис. 1. Индикаторные диаграммы скв. № 2п (пачка VII0), скв. № 1р (пачка V), скв. № 3р (пачка VII0) и скв.1э (пачка VIII) по данным исследования с 2005 по 2009 годы.

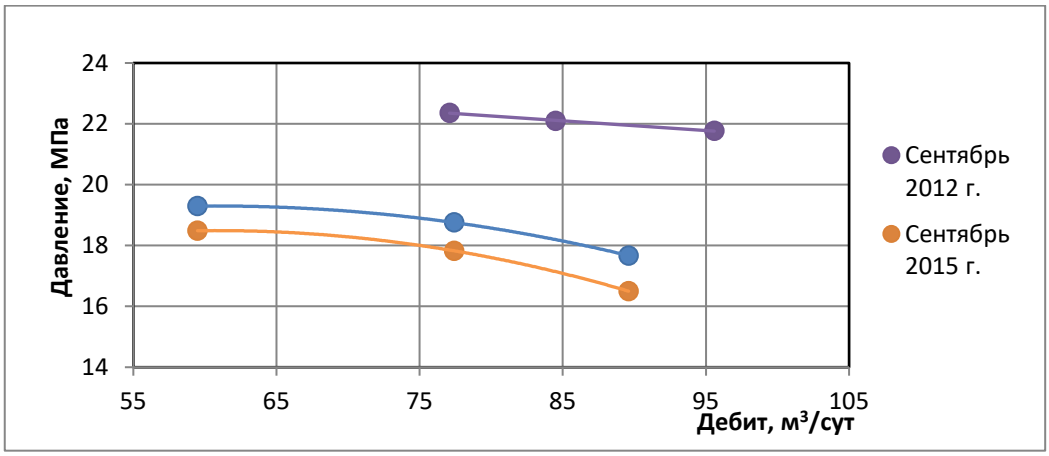

Рис. 2. Индикаторные диаграммы скв. № 2п (пачка VII0), зарегистрированные в 2012-2015 годах

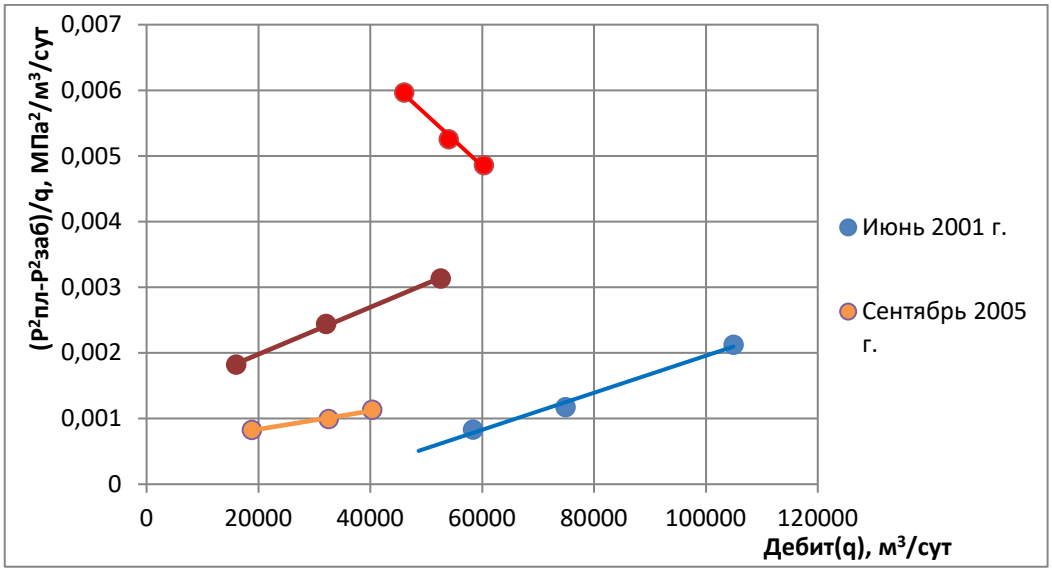

Рис. 3. Индикаторные диаграммы Скв. 1п, (газоконденсатная пачка VII).

При рассмотрении вышеприведенных исследований методом «установившихся отборов» в скважинах большинства месторождений можно констатировать, что по трем, иногда даже по четырем режимам не представляется возможным определить давление, при котором начинается снижение коэффициента продуктивности. Построенная по таким данным индикаторная диаграмма слабо представительна и соответственно достоверность рассчитанных по ней параметров работы пласта низкая.

2.2. Исследования методом регистрации кривых восстановления давления

При интерпретации данных КВД строятся графики зависимости забойного давления от времени в различных координатах. В большинстве случаев промысловыми службами выполнялась обработка данных КВД по методу «Хорнера» [1, 2, 3].

Для обработки результатов исследования методом Хорнера строится график в полулогарифмических координатах Рз2 = F(lg(Tp+t)/t)). Проводится прямая по конечным точкам графика. Экстраполированная линия до оси ординат указывает на квадрат пластового давления, в то время как наклон графика позволяет рассчитать гидропроводность пласта в районе скважины. Если на графике КВД нет прямолинейного участка, то определить фильтрационные параметры невозможно. Чаще всего отсутствие прямолинейного участка вызвано недостаточной продолжительностью регистрации КВД (от 2,5 часов) при низких ФЕС или и особенностями геологического строения пласта.

Метод Хорнера является частным случаем обобщенного дифференциального метода. Сущность обобщенного дифференциального метода (ОДМ) [4] - это прямолинейная анаморфоза кривой восстановления забойного давления или уровня, т.е. построение графика в координатах «модифицированное давление - модифицированное время», с последующим определением этом графике прямолинейного участка и вычислением по наклону графика и свободному члену ФЕС пласта. Частными случаями ОДМ обычно являются применяемые в практике методы. Так при кратковременной работе скважины с постоянным дебитом до остановки и отсутствии притока после остановки из ОДМ вытекает метод Хорнера, при длительной работе с постоянным дебитом и отсутствии притока - метод касательной, при длительной работе с постоянным дебитом и наличием притока - метод Чарного-Умрихина [1, 4].

Наиболее распространенный в настоящее время метод интерпретации КВД - метод наилучшего совмещения [4]. Согласно этому методу фактическая кривая изменения забойного давления совмещается с точным решением уравнения пьезопроводности для той или иной интерпретационной модели фильтрации в пласте. Алгоритм обработки представляет собой итерационный процесс варьирования определяемых параметров до максимально возможного совмещения фактической и расчетных кривых. Дебит аппроксимируется либо ступенчатой, либо ломаной линией.

В результате обработки данных КВД

этими методами определяются гидропроводность пласта, приведенный радиус скважины, скин-фактор, коэффициент совершенства скважины, проницаемость, пьезопроводность и пластовое давление. При этом чем точнее задается история работы и остановок скважины, тем точнее определяются параметры.

В современной практике при интерпретации полученных результатов исследований методом КВД, используются программные разработки компании Kappa Engineerring, с использованием программных продуктов, таких как «Saphir» или аналогичных [5, 6, 7, 8]. При этом появляется возможность определить не только ФЕС пласта, но и охарактеризовать модель фильтрации флюидов в пласте для охваченной исследованиями зоны.

Для обработки КВД, с использованием программы «Saphir» предварительно необходимо определить по какой интерпретационной модели следует вести обработку.

С этой целью строится диагностический график давления и его производной от времени в би-логарифмических координатах (производная Бурде) [6]. Аргументом, по которому производится дифференцирование кривой давления, является функция суперпозиции, учитывающая историю работы скважины. Диагностический график представляет собой семейство двух кривых. Одна кривая – это изменение депрессии на пласт во времени, а вторая кривая – это изменение во времени производной от депрессии. Существуют десятки характерных кривых, рассчитанных для различных типов пластов-коллекторов, для различных особенностей строения продуктивного пласта и видов фильтрации. Используя эти кривые как палетки можно определить интерпретационную модель для обрабатываемой кривой и определить все параметры, характеризующие данную модель.

Модель пласта предполагает определение строения коллектора и определение границ (непроницаемых, или границ постоянного давления), что является обязательным условием для получения достоверных результатов о фильтрационных параметрах коллектора. При использовании модели однородного неограниченного пласта определяется большой набор необходимых геолого-физических данных: гидропроводность пласта, приведенный радиус скважины, скин-фактор, коэффициент совершенства скважины, проницаемость, пьезопроводность и пластовое давление. При использовании других моделей пласта в число определяемых параметров добавляются параметры, характеризующие выбранную модель: размеры зон неоднородности, расстояния до выявленных границ в пласте, длина и проводимость искусственной трещины и т.п.

В качестве примера использования программного комплекса «Saphir» компании Kappa Engineering, можно привести результат интерпретации данных КВД, полученных при исследованиях скважин № 2п в сентябре 2012 года и по скважине № 4р от сентября 2017 года. Данные исследования одни из немногих, имеющие достаточную продолжительность регистрации КВД с большим числом зарегистрированных точек. В результате обработки данных этих КВД и их интерпретации были получены данные о фильтрационных параметрах пласта.

На рис. 4 приведен диагностический график, построенный по результатам исследований скв.4р в 2017 году.

Из рисунка видно, что в билогарифмических координатах на графике не выделяется участок радиальной фильтрации (горизонтальный отрезок параллельный оси ординат), а происходит падение графика. Такое поведение графика говорит о наличие границы постоянного давления. Скважина обводнена более чем 80% и границей постоянного давления, скорее всего, является подошвенная вода. Скважина была обработана по модели с границей постоянного давления и получено хорошее совмещение на всех графиках.

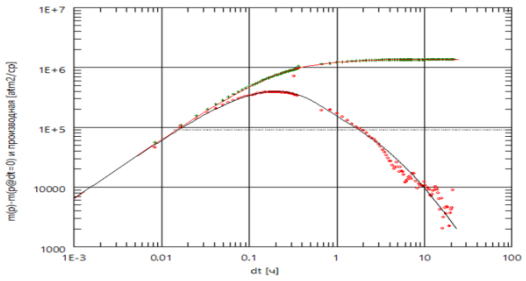

Рис. 4. Диагностический график по данным исследований скв. 4р в сентябре 2017 г.

3. Заключение

При сравнении оценок фильтрационных параметров нефтяного пласта пачки VII0, определенных по методу «Хорнера» и с использованием программы «Saphir» по данным исследования скважины № 2п, можно констатировать, что по обработке в програм ме «Saphir» было определено больше параметров, в том числе: гидропроводность, скин-фактор, выделены границы пласта (Таб. 1). По данным исследования скважины № 4р (залежь VII0) можно констатировать, что в дополнение к параметрам, определенных по

«Хорнеру» в программе «Saphir», были определены еще и границы пласта. Значения параметров, определенных по программе «Saphir», выглядят более реальными.

Таблица 1. Сравнение фильтрационных параметров, определенных методом «Хорнера» и в программе «Saphir»

Параметр	Скв. № 2п (залежь VII[0])		Скв. № 4р (залежь VII[0])	
	Метод обработки результатов КВД			
	По «Хорнеру»	В п/о «Saphir»	По «Хорнеру»	В п/о «Saphir»
Проницаемость пласта, мкм²	20,5	25,2	17,4	25,5
Скин-фактор	-	-4.75	28.04	-2.64
Пластовое давление, МПа	28.1	31.84	50.0	49.72
Определение модели фильтрации флюидов в пласте	нет	Выделены непроницаемые границы постоянного давления	нет	*Выделена граница постоянного* давления

Особенно это относится к величинам параметра проницаемости.

При анализе исследований скважин методами КВД следует отметить, что очень часто зарегистрированные кривые восстановления давления не всегда являются надлежащего качества из-за не учета работы скважины перед исследованием и отсутствия стабилизации пластового давления в процессе регистрации КВД. Поэтому при регистрации КВД необходимо придерживаться общепринятых технологий [2, 3, 9].

Одним из вариантов таких технологий можно предложить следующее. Перед регистрацией КВД скважину закрывают для спуска глубинных приборов, если перед этим она работала. После спуска глубинных приборов скважина запускается в работу на штуцере, на котором она работала перед ее остановкой. Исследования начинают после того, когда время работы скважины становится в шесть раз большим времени ее предыдущей остановки (Траб=6*Тост). Если при спуске приборов скважина не останавливалась, исследования можно начинать сразу после спуска приборов на забой. Для регистрации КВД скважину закрывают на 1-3 суток, в зависимости от параметров пласта. В процессе регистрации КВД необходимо так же регистрировать устьевые давления.

При интерпретации результатов КВД должна быть учтена история работы скважины как на режимах, так и ее работа и остановки перед исследованием.

Оценка достоверности полученных параметров в результате проводимых газогидродинамических исследований возможна только при качественном проведении самих исследований в соответствии с инструкциями и с использованием современных высокочувствительных приборов и интерпретации результатов исследований на современных программных продуктах.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН - проведение фундаментальных научных исследований по теме № 0580-2021-0019 «Создание методики выявления невыработанных зон на нефтяных месторождениях и подсчёта остаточных запасов нефти на основе комплексирования математического моделирования, анализа разработки с исследованиями скважин и пластов».

Well test procedure features identified in wells of gas-condensate reservoirs located in Krasnodar region

A.G. Dyachenko, Y.M. Steinberg, M.Y. Ahapkin

Abstract. Article describes well test results, which were conducted in wells penetradet gas-condensate saturated reservoir. Article describes features of interpretation results. Also interpretation methods results are overviewed in terms its applicability to geological conditions of saturated reservoir.

Keywords: well test results, gas-condensate reservoir

Литература

1. С.Н. Бузинов, И.Д.Умрихин. Гидродинамические методы исследования скважин и пластов. М.: Недра, 1973, 248 с.

2. Р. Эрлагер мл. Гидродинамические методы исследования скважин. М., - Ижевск: Институт компьютерных исследований, 2007, 512 с.

3. Т.А. Деева, М.Р. Камартдинов, Т.Е. Кулагина [и др.]. Гидродинамические исследования скважин: анализ и интерпретация данных. Томск: ЦППС НД ТПУ, 2009, 243 с.

4. Л.Г. Кульпин, Ю.А. Мясников. Гидродинамические методы исследования нефтегазоносных пластов. М.: Недра, 1974, 200 с.

5. Д. Бурде. Интерпретация результатов исследований скважин. Материалы лекций. М., Petroleum Engineering and Related Management Training Gubkin Academy, 1994, 109 с

6. D. Bourdet. Well Test Analysis: The Use of Advanced Interpretation Models. – Amsterdam: Elsevier, 2002, 436 p.

7. О. Узе, Витура Д., О. Фьяре. Анализ динамических потоков. Теория и практика интерпретации данных ГДИС и анализа добычи, а также использование данных стационарных глубинных манометров. - Kappa Engineerring. 2009, 359 с.

8. А.А. Колеватов, Ю.М. Штейнберг, Ю.Б. Чен-лен-сон, А.Г. Дяченко. Классификационные признаки типов нефтенасыщенных коллекторов по результатам гидродинамических исследований. «Труды. НИИСИ РАН». Т. 9 (2019), №5, 59-62.

9. РД. Газпром 086 – 2010. Инструкция по комплексным исследованиям газовых и газоконденсатных скважин. М.: Газпром экспо, 2011, Ч.I, 234 с.

Комплексная система анализа и подготовки данных по свойствам пластовых флюидов для проектных документов с использованием оперативного банка данных

К.Д. Ашмян[1], О.В. Ковалева[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kdashmyan@yandex.ru

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, olgakovaleva57@ mail.ru

Аннотация. Рассматривается система анализа и подготовки данных по свойствам пластовых флюидов для проектных документов по разработке нефтяных месторождений. Создание комплексной системы с помощью банка данных способствует более эффективному использованию исходных данных при проектировании разработки, повышает их достоверность. Это позволяет выявлять фактическое распределение физико-химических свойств пластовых нефтей по глубине и простиранию залежи, корректировать проекты разработки с учетом влияния геологических свойств залежи и техногенного влияния процесса разработки нефтяного месторождения, а также выявлять зоны распределения запасов в так называемых целиках нефти.

Ключевые слова: пластовые флюиды, системный анализ, информационный банк данных, проектирование разработки, зональное распределение физико-химических свойств нефтей в пласте

1. Введение

В настоящее время комплексной системы анализа и подготовки данных по свойствам пластовых флюидов для проектных документов нет. Это обстоятельство является серьезным недостатком при проектировании разработки. В начале разработки имеются данные, полученные в основном для подсчета запасов, т.е. до начала разработки, фактически в статическом состоянии.

При составлении проектов разработки нефтяных месторождений используется вся имеющаяся информация по физико-химическим свойствам пластовых флюидов, вне зависимости от того, каким образом и с какой целью они были получены. Отсутствие системного подхода к первичным данным приводит к тому, что принимаются некие средние значения параметров, которые не учитывают ошибки при отборе проб и их исследовании. Этот подход не имеет достаточного обоснования при анализе первичных данных, т.к. не учитывает взаимосвязей по площади и глубине залегания залежи. При таком подходе данные, полученные по одной скважине, часто могут распространяться на весь продуктивный пласт.

2. Создание комплексной системы анализа и подготовки данных

Системный анализ подготовки данных по физико-химическим свойствам основывается, во-первых, на оценке соответствия отбора и исследования глубинных и дегазированных проб нефти СТО и ОСТу [1, 2] и проведении грамотной отбраковки первичной информации. Во-вторых, на создание базы всей полученной информации, т.е. оперативного банка данных [3], позволяющего использовать данные, полученные на основе анализа и корреляционных зависимостей. Банк данных включает несколько программ формирования массива исходных данных. Обработанная информация может быть получена в виде стандартных таблиц (Рисунок 1), предназначенных для проектов разработки [4], или для построения карт распределения физико-химических свойств пластовых нефтей по площади или глубине залежей.

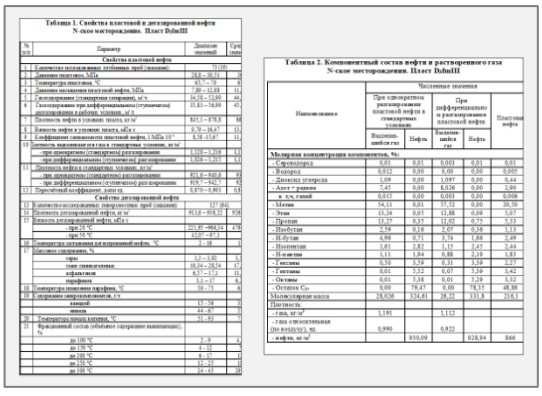

Рис. 1. Стандартные таблицы физико-химических свойств нефти N-ского месторождеия

Известно, что причины различий в свойствах нефти, проявляющиеся в процессе разработки залежи, разделяются на 2 группы:

- изменения, которые определяются природными различиями свойств нефти в различных зонах залежи. Так было выявлено зональное распределение свойств нефти по площади или глубине некоторых месторождений нефти;

- изменения свойств нефти, вызванные техногенными влиянием разработки.

В результате разработки происходит техногенное воздействие на пласт: понижение пластового давления приводит к разгазированию нефти в пластовых условиях, в т.ч. локальное разгазирование нефти в призабойной зоне скважин, образование водяных зон, внутрипластовое окисление нефти, изменение фазового состояния пластовой нефти за счет выделения газовой фазы, выделение сероводорода и др.

На нефтяных месторождениях, разрабатываемых на естественном режиме, а также на поздней стадии разработки (остаточные нефти), наблюдается уменьшение пластового давления и обводнение залежей нефти. Поэтому контроль за разработкой нефтяных месторождений и, по обратной связи [5], управление в процессе разработки качеством нефти, возможно при наличии достоверной информации и максимально полной информации о физико-химических свойствах и компонентном составе пластовых нефтей по каждому продуктивному объекту нефтяного месторождения.

На начальном периоде разработки нефтяной залежи к скважине в основном подходит более легкая нефть, потом в фильтрацию включаются более тяжелые нефти. Однако, изменение свойств нефти в процессе разработки может происходить, как в сторону утяжеления, так и в сторону облегчения добываемой нефти (например, Бакинские нефти). Опыт разработки нефтяных месторождений свидетельствует о том, что наиболее характерным случаем является утяжеление нефтей в процессе разработки.

Как уже было отмечено рядом авторов [6 - 7] плотность нефти обычно увеличивается от свода к крыльям залежи. В сводовой части залежи всегда содержится больше газа. Ближе к зонам водонефтяного контакта происходят окислительные процессы, что увеличивает плотность нефти в приконтурных зонах. Вязкость нефти увеличивается от купола свода к крыльям. Давление насыщения нефти газом и количество растворенного газа в единице объема нефти уменьшаются по направлению к водонефтяному контакту, следовательно, уменьшается и объемный коэффициент.

Большое влияние на изменение свойств нефти оказывает пластовая вода, которая за счет высокого энергетического потенциала активно участвует в вытеснении нефти из породы. Поэтому свойства вытесняемой нефти в обводненных зонах заметно отличаются от свойств нефти, полученной до начала разработки. Происходит это за счет перераспределения растворенных в нефти и пластовой воде газов и легких углеводородных компонентов [8].

При закачке в пласт воды для поддержания пластового давления происходит так же окисление нефти кислородом, растворенным в закачиваемой воде. В результате нефть становится тяжелее [9].

Используя оперативный банк данных, составленный для месторождений данного региона, можно проводить мониторинг физико-химических свойств пластовых и дегазированных нефтей по каждому продуктивному объекту и по нефтяному месторождению в целом [10].

Рассчитывая средние параметры нефти с учетом зонального распределения свойств нефти по глубине или по простиранию залежи, можно повысить точность принятых параметров нефти при подсчете запасов и при создании модели разработки.

Пример обработки данных с помощью банка данных по физико-химическим пластовых нефтей приводится на Рисунке 2.

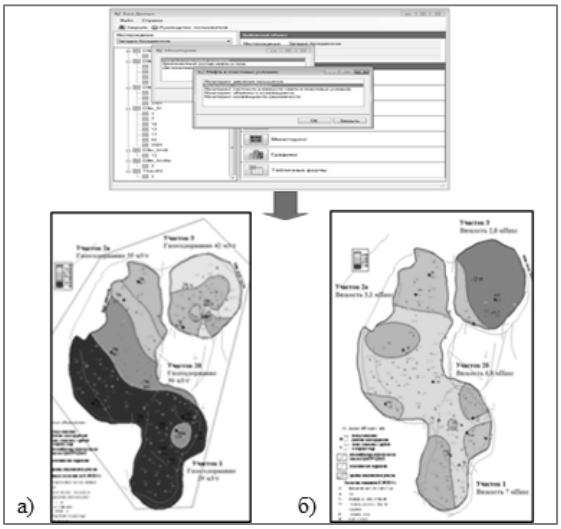

Рис. 2. Пример картопостроения распределения физико-химических свойств пластовой нефти на основе банка данных (а – газосодержания и б – вязкости пластовой нефти)

Вся информация, полученная при исследовании глубинных и поверхностных проб нефти, отобранных на данном месторождении, была занесена и протестирована в оперативном банке данных. При проведении в течение нескольких лет мониторинга и картопостроения основных параметров физико-химических свойств нефтей, была выявлена, как в пластовых, так и в поверхностных условиях, зона более легкой и менее вязкой нефти на севере залежи и зона тяжелой и нефти на юге. Так, на севере залежи средняя вязкость пластовой нефти 2,8 мПа·с, на юге – 7,0 мПа·с. Газосодержание пластовой нефти соответственно – 42 и 29 м3/т. Были выявлены также зоны с промежуточными свойствами нефти.

Для данного месторождения были рассчитаны средние параметры нефти с учетом зонального распределения свойств нефти по простиранию залежи. В результате чего, была повышена точность принятых параметров по свойствам нефти, используемых как для подсчета запасов и проектных работ, так и для эффективного применения промысловых мероприятий.

3. Заключение

Текст заключения. Системный анализ подготовки данных по физико-химическим свойствам нефти может помочь при составлении проектов разработки нефтяных месторождений для принятия решений эффективной разработки залежей нефти. Использование банка данных, позволит проводить оперативное построение карт распределения физико-химических свойств пластовых и дегазированных нефтей. Мониторинг свойств нефтей позволит осуществлять корректировки, а также применять обратную связь в действующей модели разработки [5].

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН - проведение фундаментальных научных исследований по теме № 0580-2021-0019 «Создание методики выявления невыработанных зон на нефтяных месторождениях и подсчёта остаточных запасов нефти на основе комплексирования математического моделирования, анализа разработки с исследованиями скважин и пластов».

A Comprehensive System for Analyzing and Preparing Data on the Properties of Reservoir Fluids for Project Documents Using an Operational Data Bank

K. D. Ashmyan, O. V. Kovaleva

Abstract. The system of analysis and preparation of data on the properties of reservoir fluids for project documents for the development of oil fields is considered. The creation of a comprehensive system using a data bank contributes to a more efficient use of source data in the design of development, to increase their reliability. This makes it possible to identify the actual distribution of the physical and chemical properties of reservoir oils over the depth and strike of the deposit, to adjust development projects taking into account the influence of the geological properties of the deposit and the technogenic influence of the oil field development process, as well as to identify the zones of distribution of oil reserves in the so-called.

Keywords: reservoir fluids, system analysis, information database, development design, zonal distribution of physical and chemical properties of oil in the reservoir

Литература

1. СТО РМНТК 153-39.2-002-2003. Нефть. Отбор проб пластовых флюидов.
2. ОСТ 153-39.2-048-2003. Нефть. Типовое исследование пластовых флюидов и сепарированных нефтей.
3. К.Д. Ашмян, О.В. Ковалева, Д.О. Казаков. Создание региональных оперативных банков данных по свойствам нефтей. Сб. научных трудов «Повышение эффективности разработки нефтяных месторождений», ОАО «ВНИИнефть», №148, 2013, 65 – 72.
4. Методические рекомендации по применению классификации запасов и ресурсов нефти и горючих газов. Утверждено распоряжением Минприроды России от 01.02.2016г., №3-р.
5. К.Д. Ашмян, С.Г. Вольпин, О.В. Ковалева, Г.А. Ковалева. Создание системы управления с обратной связью при разработке нефтяных месторождений. «Труды НИИСИ», Т. 10 (2020),

№ 3, 34–38.
6. И.М. Амерханов. Закономерности изменения свойств пластовых жидкостей при разработке нефтяных месторождений. Обзорная информация. Серия «Нефтепромысловое дело», М., ВНИИОЭНГ, 1980. 48 с.
7. А.И. Хазнаферов. Исследование пластовых нефтей. М, Недра, 1987, 116 с.
8. А.Ю. Намиот. Фазовое равновесие в добычи нефти. М., Недра, 1976, 183 с.
9. А.Г. Козлов, О.В. Ковалева. Окислительные процессы в нефтях различных месторождений. Сб. научных трудов «Изучение особенностей применения методов повышения нефтеотдачи», ВНИИнефть., № 102, 1988, 88-93.
10. О.В. Ковалева, В.И. Джафарова, К.Д. Ашмян. Закономерности распределения начальных свойств нефти на основе анализа данных «Оперативного банка по физико-химическим свойствам нефти» и картопостроения на примере двух месторождений. Сб. научных трудов «Повышение эффективности разработки месторождений с трудноизвлекаемыми запасами нефти», ОАО «ВНИИнефть», №153, 2015, 47 – 64.

Алгоритмы определения коллизий аппроксимирующих параллелепипедов с моделью рельефа местности

Е.В. Страшнов[1], Д.В. Омельченко[2]

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

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

Аннотация. В работе рассматривается задача определения коллизий аппроксимирующих параллелепипедов (боксов), окружающих геометрию виртуальных объектов, с рельефом местности, заданным в виде полигональной сетки. Для решения этой задачи предлагаются алгоритмы, основанные на применении теоремы о разделяющей оси, геометрических тестах, а также методах отсечения отрезков и многоугольников. В этих алгоритмах вычисление контактной информации о пересечении осуществляется методом отложенной обработки треугольников с сортировкой глубин проникновения. Такой подход позволяет эффективно определить те треугольники полигональной сетки, которые образуют границу пятна контакта бокса с рельефом местности. Апробация разработанных алгоритмов проводилась в программном комплексе виртуального окружения на примере взаимодействия виртуальных объектов в форме бокса с рельефом местности различного типа.

Ключевые слова: определение коллизий, аппроксимирующий параллелепипед (бокс), рельеф местности, полигональная сетка, контактное многообразие, теорема о разделяющей оси, отсечение, система виртуального окружения

1. Введение

Существует ряд областей исследований, в которых требуется осуществлять компьютерное моделирование движения объектов в виртуальном окружении. Примером служит робототехника, в рамках которой создаются имитационно-тренажерные комплексы, предназначенные для тренировки операторов навыкам управления колесными и гусеничными роботами, манипуляторами, летательными аппаратами и т.д. В процессе моделирования виртуальные объекты могут взаимодействовать (сталкиваться) друг с другом, что приводит к задаче определения пересечений (коллизий) [1], [2] этих объектов. При этом разрабатываемые алгоритмы для решения этой задачи должны обеспечивать моделирование движения виртуальных объектов в масштабе реального времени с обеспечением требования физической правдоподобности их поведения при столкновении. Разработка таких алгоритмов является важной и актуальной задачей.

В данной работе рассматривается задача определения коллизий аппроксимирующих параллелепипедов (боксов) [3], окружающих геометрию виртуальных объектов, с моделью рельефа местности, который задан в виде полигональной сетки, состоящей из множества связанных треугольников. Один из подходов для решения этой задачи заключается в том, чтобы представить рельеф местности в виде набора выпуклых объектов. Тогда задача сводится к определению коллизий между боксом и каждым объектом полигональной сетки. Существующие решения этой задачи включают семейство алгоритмов GJK-EPA [4], [5], [6], [7] и алгоритмы, основанные на применении теоремы о разделяющей оси SAT [8], [9] (*англ.* Separating Axis Theorem). Однако при таком подходе для того, чтобы полностью окружить рельеф местности, в общем случае требуется большое число объектов, что затрудняет реализацию алгоритмов в реальном времени. Поэтому существует другой подход, в рамках которого бокс, как и рельеф местности, задается в виде полигональной сетки. Тогда пересечение бокса с рельефом рассматривается относительно треугольников двух геометрий [10], [11]. Как правило, при таком подходе количество тестов на пересечение является избыточным и для произвольного рельефа сложно правильно сформировать контактное множество. Альтернативное решение заключается в определении коллизий бокса, заданного в аналитическом виде, с треугольниками полигональной сетки [12], [13]. Такой подход рассмотрен в настоящей работе с представлением рельефа в виде регулярной полигональной модели на основе сетки высот. Предлагаемое решение включает реализацию алгоритмов, в которых определение пересечения бокса с треугольниками полигональной сетки осуществляется с применением теоремы о разделяющей оси, а вычисление контактной информации выполняется с помощью эффективных

геометрических тестов. При этом для ускорения расчетов и правильной аппроксимации пятна контакта предлагается подход, в котором осуществляется отложенная обработка треугольников. Это позволяет правильно определить те треугольники, которые находятся на границе пятна контакта бокса с полигональной сеткой. Предложенные в статье алгоритмы были реализованы в программном комплексе виртуального окружения, разработанном в ФГУ ФНЦ НИИСИ РАН, и апробированы на примере взаимодействия объектов, имеющих форму бокса, со сложным рельефом местности.

2. Постановка задачи

Рассмотрим аппроксимирующий бокс (см. рис. 1), который окружает геометрию виртуального объекта и имеет размеры (полудлины) *l*, *w* и *h*. Локальная система координат бокса находится в его центре, имеет начало в точке O_B и оси, заданные единичными ортами $\mathbf{a}_i$, $i = 1,2,3$. Вершинами бокса являются точки Q_j, $j = \overline{1,8}$.

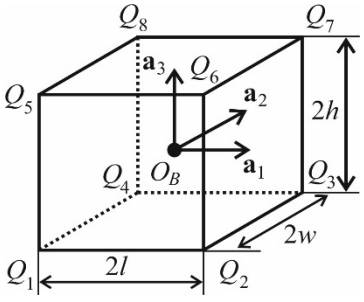

Рис. 1. Аппроксимирующий бокс

Пусть рельеф местности задается в виде полигональной сетки. Тогда с помощью алгоритма Моллера-Трумбора [14] выполняется построение регулярной прямоугольной сетки, в узлах которой вычисляются значения высот. Преимущество такого подхода заключается в эффективном доступе к координатам вершин и независимости от детализации рельефа. С подробностями реализации алгоритма построения регулярной сетки высот можно ознакомиться в работе [15]. Для этой сетки высот будем рассматривать полигональную модель, проекция которой на горизонтальную плоскость показана на рис. 2. Координаты вершин полигональной сетки в мировой системе координат вычисляются через индексы *i*, *j* и значения двумерного массива высот h_{ij}:

$$V_{ij} = \left(x_0 + i\Delta L,\, y_0 + j\Delta L,\, h_{ij}\right),$$

где (x_0, y_0) – начальная точка прямоугольной сетки, ΔL – шаг сетки.

Задача определения коллизий аппроксимирующего бокса с рельефом местности состоит в проверке самого факта пересечения и вычислении необходимой контактной информации. Пересечение аппроксимирующего бокса с рельефом задается в виде *контактного многообразия* (*англ.* contact manifold), состоящего из набора параметров $C_P = \{P, \mathbf{n}, d\}$, где P – точка контакта на поверхности рельефа, $\mathbf{n}$ – единичный вектор нормали в точке P, d – глубина проникновения.

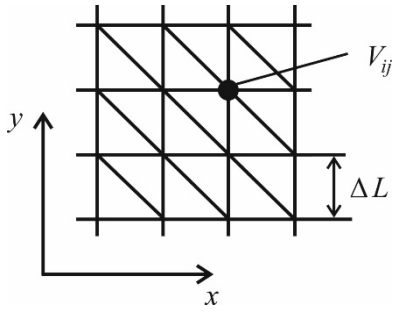

Рис. 2. Регулярная полигональная сетка

Предлагаемое решение этой задачи состоит из нескольких этапов, куда входит широкая и узкая фаза определения коллизий бокса с рельефом, а также построение контактного многообразия. Широкая фаза включает в себя быстрые тесты на пересечение с определением множества треугольников, с которыми потенциально может пересекаться бокс. На узкой стадии для каждого из полученных треугольников выполняется проверка на пересечение. В том случае, если бокс пересекается с треугольником, то для этого треугольника вычисляются контактные точки. Наконец на последней стадии осуществляется отбор контактных точек, которые наиболее эффективно аппроксимируют пятно контакта бокса с рельефом. Далее подробно опишем предлагаемые алгоритмы.

3. Широкая фаза определения коллизий бокса с рельефом

На этой стадии выполняется грубая проверка на пересечение, в которой вычисляются границы $[x_{\min}, x_{\max}]$ и $[y_{\min}, y_{\max}]$ проекции бокса на горизонтальную плоскость, где $x_{\min} = \min Q_{jx}$, $x_{\max} = \max Q_{jx}$, $y_{\min} = \min Q_{jy}$, $y_{\max} = \max Q_{jy}$, $j = \overline{1,8}$. По этой границе определяется множество Ω тех треугольников полигональной сетки, с которыми потенциально может пересекаться бокс. Кроме того, дополнительно вычисляются нижняя граница $z_{\min} = \min Q_{jz}$, $j = \overline{1,8}$ бокса по оси *z*, а также минимальное $h_{\min}$ и максимальное $h_{\max}$ значение сетки высот для полученного мно-

жества Ω. Тогда, если $z_{\min} > h_{\max}$, то бокс не пересекается с рельефом местности.

В противном случае на этой стадии проверяется, является ли участок поверхности рельефа горизонтальной плоскостью. Это соответствует условию $|h_{\max} - h_{\min}| \le \varepsilon$, где ε – заданная точность. Если это условие выполнено, то коллизия бокса с рельефом сводится к проверке пересечения между боксом и плоскостью с нормалью $\mathbf{n}_P = (0,0,1)^{\mathrm{T}}$. Для этого сначала вычисляется вершина бокса, которая наиболее удалена в противоположном направлении от нормали:

$$Q^{(1)} = O_B - \mathrm{sgn}(a_{1z}) l \mathbf{a}_1 - \mathrm{sgn}(a_{2z}) w \mathbf{a}_2 - \mathrm{sgn}(a_{3z}) h \mathbf{a}_3 . \quad (1)$$

Глубина проникновения точки $Q^{(1)}$ относительно плоскости равна

$$d_1 = h_{\min} - Q_z^{(1)} . \quad (2)$$

Если $d_1 \ge 0$, то бокс и плоскость пересекаются. Покажем, как осуществляется вычисление контактных точек на примере случая, когда выполнено $l|a_{1z}| \le w|a_{2z}| \le h|a_{3z}|$ (первая ось бокса имеет минимальную проекцию вдоль нормали). Тогда определение коллизий бокса с плоскостью реализуется с помощью *Алгоритма 1*.

1. Вычисляем вершину бокса $Q^{(1)}$ из (1).

2. Вычисляем глубину d_1 по формуле (2).

3. Если $d_1 < 0$, то бокс и плоскость не пересекаются. В противном случае:

3.1. Формируем первую точку контакта $C_P^1 = \{P_1, \mathbf{n}_P, d_1\}$, где $P_1 = Q^{(1)} + d_1 \mathbf{n}_P$.

3.2. Вычисляем $d_2 = d_1 - 2l|a_{1z}|$.

3.3. Если $d_2 \ge 0$, то

3.3.1. $Q^{(2)} = Q^{(1)} + 2\,\mathrm{sgn}(a_{1z}) l \mathbf{a}_1$;

3.3.2. формируем вторую точку контакта $C_P^2 = \{P_2, \mathbf{n}_P, d_2\}$, где $P_2 = Q^{(2)} + d_2 \mathbf{n}_P$.

3.3.3. вычисляем $d_3 = d_1 - 2w|a_{2z}|$.

3.3.4. если $d_3 \ge 0$, то

- $Q^{(3)} = Q^{(1)} + 2\,\mathrm{sgn}(a_{2z}) w \mathbf{a}_2$;

- формируем третью точку контакта $C_P^3 = \{P_3, \mathbf{n}_P, d_3\}$, где $P_3 = Q^{(3)} + d_3 \mathbf{n}_P$.

3.4. Вычисляем $d_4 = d_2 + d_3 - d_1$.

3.5. Если $d_4 \ge 0$, то формируем четвертую точку контакта $C_P^4 = \{P_4, \mathbf{n}_P, d_4\}$, где $P_4 = P_2 + P_3 - P_1$.

Конец алгоритма.

Алгоритм 1. Определение коллизий бокса с плоскостью.

В этом алгоритме контактные точки вычисляются для вершин бокса, которые располагаются ниже плоскости. Максимально возможно четыре точки при контакте бокса с гранью. Случаи, когда вторая и третья оси имеют минимальные проекции вдоль нормали, обрабатываются аналогичным образом. Если участок поверхности не является горизонтальной плоскостью, то проверка на пересечение осуществляется для каждого треугольника множества Ω.

4. Узкая фаза определения коллизий бокса с рельефом

На этой стадии выполняется проверка на пересечение бокса с каждым треугольником множества Ω. Задача заключается в том, чтобы определить те треугольники, с которыми пересекается бокс, и в случае пересечения вычислить контактные точки. Далее опишем предлагаемые решения этой задачи.

4.1. Определение пересечения бокса и треугольника с помощью теоремы о разделяющей оси

Для определения коллизий бокса с треугольником воспользуемся теоремой о разделяющей оси (см. [8], [9]). Согласно этой теореме два выпуклых объекта не пересекаются, если найдется плоскость, которая их разделяет. Другими словами, это соответствует тому, что не пересекаются проекции объектов на ось, которая является перпендикуляром к разделяющей плоскости. В том случае, если такая ось не найдена, то выпуклые объекты пересекаются.

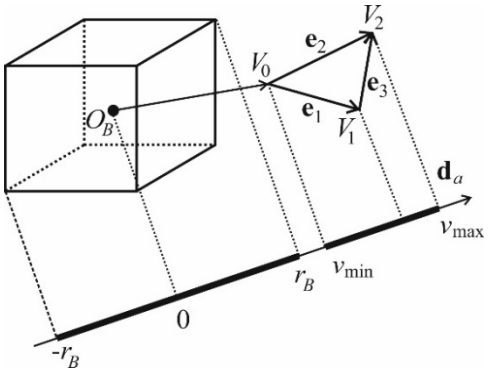

Рис. 3. Проекции бокса и треугольника на разделяющую ось

Пусть треугольник имеет вершины V_k, $k = \overline{0,2}$. Обозначим векторы ребер треугольника через $\mathbf{e}_1 = \mathbf{V}_0\mathbf{V}_1$, $\mathbf{e}_2 = \mathbf{V}_0\mathbf{V}_2$ и $\mathbf{e}_3 = \mathbf{V}_1\mathbf{V}_2$. Тогда нормаль к плоскости треугольника вычисляется как $\mathbf{n}_P = (\mathbf{e}_1 \times \mathbf{e}_2) / \|\mathbf{e}_1 \times \mathbf{e}_2\|$.

Рассмотрим проекции бокса и треугольника на разделяющую ось $\mathbf{d}_a$, как показано на рис. 3.

Проецирование будем выполнять относительно центра бокса O_B. В силу симметрии бокса величина

$$r_B = l\left|\mathbf{a}_1 \cdot \mathbf{d}_a\right| + w\left|\mathbf{a}_2 \cdot \mathbf{d}_a\right| + h\left|\mathbf{a}_3 \cdot \mathbf{d}_a\right|$$

задает отрезок $\left[-r_B, r_B\right]$ проекции бокса на разделяющую ось.

В свою очередь, проекция треугольника на разделяющую ось определяется величинами

$$\begin{aligned} v_{\min} &= \min\left\{s_{v0}, s_{v1}, s_{v2}\right\}, \\ v_{\max} &= \max\left\{s_{v0}, s_{v1}, s_{v2}\right\}, \end{aligned} \qquad (3)$$

где $s_{vk} = \mathbf{O}_B\mathbf{V}_k \cdot \mathbf{d}_a$, $k = \overline{0,2}$.

Известно [9], что для выпуклых многогранников разделяющими осями могут быть только нормали к их граням и попарные векторные произведения их ребер. Тогда для определения коллизий бокса с треугольником требуется 13 разделяющих осей (нормаль к треугольнику, 3 оси бокса и 9 векторных произведений ребра бокса и ребра треугольника). В таблице 1 приведены величины r_B и s_{vk} при проецировании бокса и треугольника на эти разделяющие оси.

Если найдется такая ось $\mathbf{d}_a$, для которой будет выполнено $v_{\min} > r_B$ или $v_{\max} < -r_B$, то бокс и треугольник не пересекаются. В противном случае, в процессе вычислений сохраняем номер *index* разделяющей оси, вектор оси **n** и глубину

$$d_{\min} = \min\left\{r_B - v_{\min}, r_B + v_{\max}\right\},$$

для которых пересечение бокса и треугольника минимально. Определение коллизий бокса и треугольника реализуется с помощью *Алгоритма 2*.

1. Инициализация: *index* = 0, $\mathbf{n} = \left(0,0,1\right)^{\mathrm{T}}$, $d_{\min} = D^{\max}$, где $D^{\max} > 0$ – большое значение.

2. Цикл $i = \overline{1,13}$ по разделяющим осям:

2.1. Вычисляем $\mathbf{d}_a^i$;

2.2. Если $\left\|\mathbf{d}_a^i\right\| \le \delta$ (ось вырождена), то переходим к следующей оси, где δ – заданное значение точности;

2.3. Вычисляем r_B^i, s_{vk}^i по таблице 1, где $k = \overline{0,2}$;

2.4. Определяем $v_{\min}^i$ и $v_{\max}^i$ из (3);

2.5. Если $v_{\min}^i > r_B^i$ или $v_{\max}^i < -r_B^i$, то бокс и треугольник не пересекаются. Досрочный выход из цикла;

2.6. Если $\mathbf{d}_a^i \cdot \mathbf{n}_P \le 0$, то $\mathbf{d}_a^i = -\mathbf{d}_a^i$;

2.7. Вычисляем

$$d_i = \min\left\{r_B^i - v_{\min}^i, r_B^i + v_{\max}^i\right\};$$

2.8. Если $d_i < d_{\min}$, то *index* = *i*, $\mathbf{n} = \mathbf{d}_a^i$, $d_{\min} = d_i$.

Конец алгоритма.

Алгоритм 2. Определение коллизий бокса с треугольником.

Таблица 1. Величины проекций бокса и треугольника на разделяющие оси

N	$\mathbf{d}_a$	r_B	s_{v0}	s_{v1}	s_{v2}
1	$\mathbf{n}_P$	$l\left\|\mathbf{a}_1 \cdot \mathbf{n}_P\right\| + w\left\|\mathbf{a}_2 \cdot \mathbf{n}_P\right\| + h\left\|\mathbf{a}_3 \cdot \mathbf{n}_P\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot \mathbf{n}_P$	s_{v0}	s_{v0}
2	$\mathbf{a}_1$	l	$\mathbf{O}_B\mathbf{V}_0 \cdot \mathbf{a}_1$	$s_{v0} + \mathbf{e}_1 \cdot \mathbf{a}_1$	$s_{v0} + \mathbf{e}_2 \cdot \mathbf{a}_1$
3	$\mathbf{a}_2$	w	$\mathbf{O}_B\mathbf{V}_0 \cdot \mathbf{a}_2$	$s_{v0} + \mathbf{e}_1 \cdot \mathbf{a}_2$	$s_{v0} + \mathbf{e}_2 \cdot \mathbf{a}_2$
4	$\mathbf{a}_3$	h	$\mathbf{O}_B\mathbf{V}_0 \cdot \mathbf{a}_3$	$s_{v0} + \mathbf{e}_1 \cdot \mathbf{a}_3$	$s_{v0} + \mathbf{e}_2 \cdot \mathbf{a}_3$
5	$\mathbf{a}_1 \times \mathbf{e}_1$	$w\left\|\mathbf{a}_3 \cdot \mathbf{e}_1\right\| + h\left\|\mathbf{a}_2 \cdot \mathbf{e}_1\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_1 \times \mathbf{e}_1)$	s_{v0}	$s_{v0} + (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_1$
6	$\mathbf{a}_1 \times \mathbf{e}_2$	$w\left\|\mathbf{a}_3 \cdot \mathbf{e}_2\right\| + h\left\|\mathbf{a}_2 \cdot \mathbf{e}_2\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_1 \times \mathbf{e}_2)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_1$	s_{v0}
7	$\mathbf{a}_1 \times \mathbf{e}_3$	$w\left\|\mathbf{a}_3 \cdot \mathbf{e}_3\right\| + h\left\|\mathbf{a}_2 \cdot \mathbf{e}_3\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_1 \times \mathbf{e}_3)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_1$	s_{v1}
8	$\mathbf{a}_2 \times \mathbf{e}_1$	$l\left\|\mathbf{a}_3 \cdot \mathbf{e}_1\right\| + h\left\|\mathbf{a}_1 \cdot \mathbf{e}_1\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_2 \times \mathbf{e}_1)$	s_{v0}	$s_{v0} + (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_2$
9	$\mathbf{a}_2 \times \mathbf{e}_2$	$l\left\|\mathbf{a}_3 \cdot \mathbf{e}_2\right\| + h\left\|\mathbf{a}_1 \cdot \mathbf{e}_2\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_2 \times \mathbf{e}_2)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_2$	s_{v0}
10	$\mathbf{a}_2 \times \mathbf{e}_3$	$l\left\|\mathbf{a}_3 \cdot \mathbf{e}_3\right\| + h\left\|\mathbf{a}_1 \cdot \mathbf{e}_3\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_2 \times \mathbf{e}_3)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_2$	s_{v1}
11	$\mathbf{a}_3 \times \mathbf{e}_1$	$l\left\|\mathbf{a}_2 \cdot \mathbf{e}_1\right\| + w\left\|\mathbf{a}_1 \cdot \mathbf{e}_1\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_3 \times \mathbf{e}_1)$	s_{v0}	$s_{v0} + (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_3$
12	$\mathbf{a}_3 \times \mathbf{e}_2$	$l\left\|\mathbf{a}_2 \cdot \mathbf{e}_2\right\| + w\left\|\mathbf{a}_1 \cdot \mathbf{e}_2\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_3 \times \mathbf{e}_2)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_3$	s_{v0}
13	$\mathbf{a}_3 \times \mathbf{e}_3$	$l\left\|\mathbf{a}_2 \cdot \mathbf{e}_3\right\| + w\left\|\mathbf{a}_1 \cdot \mathbf{e}_3\right\|$	$\mathbf{O}_B\mathbf{V}_0 \cdot (\mathbf{a}_3 \times \mathbf{e}_3)$	$s_{v0} - (\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{a}_3$	s_{v1}

Выходом этого алгоритма является определение самого факта пересечения бокса и треугольника, а также направление **n**, при котором глубина $d_{\min}$ минимальна. При вычислении рассматриваются только невырожденные разделяющие оси, которые находятся в той же полуплоскости, что и нормаль $\mathbf{n}_P$ (условие $\mathbf{d}_a \cdot \mathbf{n}_P > 0$).

4.2. Вычисление потенциальных контактных точек

В том случае, если бокс и треугольник пересекаются, то требуется вычислить потенциальные контактные точки, которые формируют контактное многообразие пересечения бокса с рельефом местности.

Предлагаемое решение этой задачи состоит в том, что сначала рассматриваются те вершины бокса, которые находятся ниже плоскости треугольника. Если проекция вершины бокса попадает внутрь треугольника, то эта проекция является потенциальной точкой контакта. Для проверки воспользуемся алгоритмом, который описан в [1]. Пусть P – проекция вершины бокса Q на плоскость треугольника. Рассмотрим векторы $\mathbf{u} = \mathbf{PV}_1 \times \mathbf{PV}_2$, $\mathbf{v} = \mathbf{PV}_2 \times \mathbf{PV}_0$ и $\mathbf{w} = \mathbf{PV}_0 \times \mathbf{PV}_1$. Тогда точка P находится внутри треугольника, если выполнены условия

$$\mathbf{u} \cdot \mathbf{v} \ge 0,\ \mathbf{v} \cdot \mathbf{w} \ge 0 \text{ и } \mathbf{u} \cdot \mathbf{w} \ge 0,$$

а $C_P = \{P, \mathbf{n}_P, d\}$ является потенциальной точкой контакта, где $d = \mathbf{QV}_0 \cdot \mathbf{n}_P$.

Если не найдена вершина бокса, которая находится ниже плоскости треугольника, для которой эти условия выполнены, то далее переходим к следующему шагу. На этом шаге выполняется проверка на пересечение треугольника с ребрами бокса, которые находятся ниже плоскости. Для этого воспользуемся алгоритмом Кируса-Бека [16] отсечения отрезка многоугольником. Здесь в качестве отрезка рассматривается проекция $\mathbf{Q}_1'\mathbf{Q}_2'$ ребра бокса на плоскость треугольника. На выходе этого алгоритма получим параметры t_1 и t_2 отсечения отрезка треугольником. Если $t_i \in [0,1]$, то отрезок пересекает треугольник, а $C_P^i = \{P_i, \mathbf{n}_P, d_i\}$ являются потенциальными точками контакта, где $P_i = Q_1' + t_i \mathbf{Q}_1'\mathbf{Q}_2'$, $d_i = (\mathbf{Q}_1\mathbf{V}_0 - t_i \mathbf{Q}_1\mathbf{Q}_2) \cdot \mathbf{n}_P$, $i = 1,2$.

Наконец, если не найдено ребро бокса, проекция которого пересекает треугольник, то контактные точки вычисляем на основе информации, полученной при проверке на пересечение бокса с треугольником (см. *Алгоритм 2*). Всего рассматриваются три случая, в которых разделяющая ось является нормалью к плоскости треугольника, осью бокса или векторным произведением ребра бокса с ребром треугольника.

Если разделяющая ось является нормалью к плоскости треугольника, то для поиска контактных точек рассматривается грань бокса с вершинами $Q^{(k)}$, $k = \overline{1,4}$, наиболее удаленная в противоположном направлении относительно нормали. Затем выполняется отсечение этой грани бокса 4-мя плоскостями треугольника: плоскостью грани треугольника и тремя плоскостями, которые образуются ребром и нормалью к плоскости треугольника. На выходе получим многоугольник с вершинами $S^{(k)}$, $k = \overline{1,K}$, где K – число вершин. Тогда контактными точками $C_P^k = \{P_k, \mathbf{n}_P, d_k\}$ являются проекции вершин многоугольника на плоскость треугольника, где $d_k = \mathbf{S}^{(k)}\mathbf{V}_0 \cdot \mathbf{n}_P$, $P_k = S^{(k)} + d_K \mathbf{n}_P$.

В случае, если разделяющая ось совпадает с одной из осей бокса, то для поиска контактных точек выполняется отсечение треугольника гранями бокса. После отсечения получим многоугольник $V^{(j)}$, $j = \overline{1,M}$, где M – число вершин. Вершины полученного многоугольника образуют потенциальные контактные точки $C_P^j = \{V^{(j)}, \mathbf{n}, d_j\}$, где на примере $\mathbf{n} = \mathbf{a}_1$ получим, что $d_j = l - \mathbf{O}_B\mathbf{V}^{(j)} \cdot \mathbf{n}$.

Последний случай соответствует тому, что разделяющая ось является векторным произведение ребра бокса с ребром треугольника. В этом случае контактная точка лежит на общем перпендикуляре к этим ребрам. На примере ребра треугольника с вектором $\mathbf{e}_1$, получим, что $C_P = \{P, \mathbf{n}, d_{\min}\}$ является контактной точкой, где $P = V_0 + \alpha \mathbf{e}_1$, а α – параметр, с вычислением которого можно ознакомиться в публикации [17].

4.3. Вычисление границы пятна контакта

В общем случае определение коллизий бокса с полигональной сеткой сводится к тому, что на выходе получается большое число треугольников, с которыми пересекается бокс. Эти треугольники образуют пятно контакта бокса с рельефом местности. Задача заключается в том, чтобы найти те треугольники, которые находятся на границе пятна контакта. Пусть треугольники, с которыми пересекается бокс, добавляются в список треугольников L_S, упорядоченный по убыванию глубин проникновения. Обрабатывая последовательно этот список, будем заполнять множество Ψ_S вершин пятна контакта и множество Ψ_δ вершин его границы. При заполнении для каждой вершины будем вычислять число f_v

треугольников, к которым принадлежит эта вершина.

Идея вычисления границы пятна контакта бокса с полигональной сеткой заключается в следующем. Рассмотрим треугольник $\Delta \in L_S$ с вершинами V_k, $k = \overline{0,2}$. Если $V_k \notin \Psi_S$, то вершина добавляется в оба множества Ψ_S и Ψ_δ с $f_v = 1$. В том случае, если $V_k \in \Psi_S$, то число f_v увеличивается на единицу. Если вершина входит в три треугольника и более ($f_v \geq 3$), то она удаляется из множества Ψ_δ. На выходе алгоритма получим аппроксимацию пятна контакта в виде многоугольника с вершинами $V^{(r)}$, $r = \overline{1,R}$. Иллюстрация предлагаемого подхода показана на рис. 4. В этом примере для пятна контакта получено шесть вершин многоугольника.

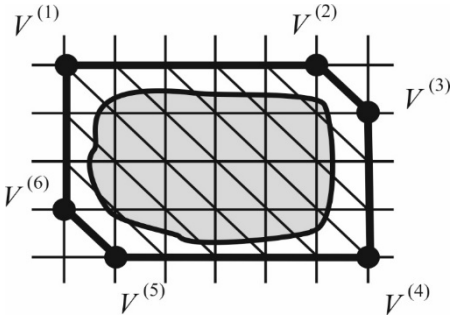

Рис. 4. Аппроксимация пятна контакта

Полученные вершины аппроксимации пятна контакта принадлежат треугольникам, для которых вычисляются потенциальные точки контакта C_P и добавляются в список $L(C_P)$. Приведенный подход для вычисления контактных точек на границе пятна контакта реализуется с помощью *Алгоритма 3*.

1. Цикл по треугольникам Δ_i, $i = \overline{1,N}$ из множества Ω:

1.1. Если бокс пересекается с треугольником Δ_i, то

- вычисляем потенциальные контактные точки C_P^s, где $s = \overline{1,S}$;

- вычисляем глубину d_i проникновения бокса и треугольника;

- добавляем треугольник Δ_i в список L_S с сортировкой по d_i.

2. Цикл по треугольникам Δ_j, $j = \overline{1,M}$ из списка L_S:

2.1. Цикл по вершинам V_k^j, $k = \overline{0,2}$ треугольника Δ_j:

2.1.1. Если $V_k^j \notin \Psi_S$, то

- добавляем V_k^j в Ψ_S и Ψ_δ;

- $f_v^j = 1$;

- для вершины V_k^j сохраняем ссылку на треугольник Δ_j,

в противном случае $f_v^j = f_v^j + 1$.

2.1.2. Если $f_v^j \geq 3$, то удаляем V_k^j из Ψ_δ.

3. Цикл по вершинам V_r, $r = \overline{1,R}$ из множества Ψ_δ:

3.1. извлекаем треугольник Δ_r: $V_r \in \Delta_r$;

3.2. добавляем контактные точки C_P^s в список $L(C_P)$, где $s = \overline{1,S}$.

Конец алгоритма.

Алгоритм 3. Вычисление контактных точек на границе пятна контакта бокса с полигональной сеткой.

Полученный список контактных точек $L(C_P)$ сокращается до минимального числа, необходимого на стадии разрешения коллизий аппроксимирующего бокса с рельефом местности. С подробностями реализации сокращения списка контактных точек можно ознакомиться в работе [18].

5. Результаты моделирования

Предложенные в статье алгоритмы определения коллизий аппроксимирующих боксов с рельефом местности были реализованы в программном комплексе виртуального окружения [19], созданном в ФГУ ФНЦ НИИСИ РАН. Этот программный комплекс состоит из подсистем управления, динамики и визуализации. Подсистема управления предназначена для управления движением виртуальными объектами, куда входят колесные и гусеничные роботы, манипуляторы, лифты, камеры, источники освещения и т.д. В этой подсистеме выполняется расчет управляющих сигналов (например, напряжений, подаваемых на электрические двигатели робота), которые поступают в подсистему динамики. В свою очередь, подсистема динамики осуществляет расчет новых координат виртуальных объектов с учетом их динамических характеристик. Эта подсистема включает модуль расчета определения коллизий виртуальных объектов, куда входит широкая и узкая фаза, а также построение контактного многообразия с вычислением контактных точек, которые используются для разрешения коллизий объектов на основе импульсов. Новые координаты объектов поступают в подсистему визуализации, которая

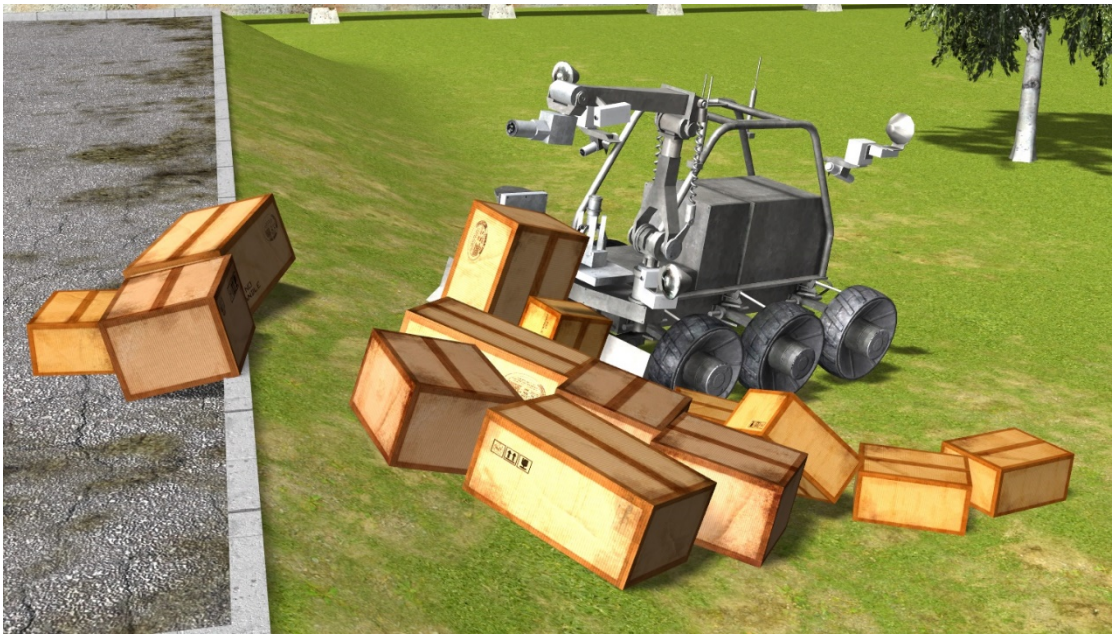

Рис. 5. Моделирование разбора завала с помощью робота КРТ-200

выполняет высококачественный рендеринг виртуальной сцены в масштабе реального времени.

Апробация разработанных алгоритмов была проведена в виртуальной сцене с участком рельефа местности размером 500 метров с шагом сетки высот $\Delta L = 4$ см. Для реализации алгоритмов были выбраны следующие параметры: $\varepsilon = 0.01$ см, $\delta = 10^{-6}$, $D^{\max} = 10^{8}$ см. Моделирование проводилось на примере выполнения роботом КРТ-200 операции разбора завала из коробок (см. рис. 5). Рассматриваемый пример включает несколько случаев контакта боксов с рельефом местности различной сложности: от ровного асфальта до холмистой поверхности. Результаты апробации показали, что разработанные алгоритмы позволяют адекватно и эффективно моделировать определение коллизий аппроксимирующих боксов со сложным рельефом местности в масштабе реального времени.

6. Заключение

В данной работе рассмотрена задача определения коллизий аппроксимирующих цилиндров с виртуальным рельефом местности, заданным в виде регулярной полигональной сетки. Для ее решения разработаны алгоритмы, которые основаны на эффективных геометрических тестах, широко применяемые в компьютерной графике. Полученные в работе результаты могут быть использованы для решения многих практических задач в системах виртуального окружения, имитационно-тренажерных комплексах, образовательных приложениях, анимации, компьютерных играх и т.д.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00387.

Collision Detection Algorithms of Bounding Boxes with Terrain Model

Evgeny Strashnov, Denis Omelchenko

Abstract. The paper considers the problem of collision detection of bounding parallelepipeds (boxes) surrounding the geometry of virtual objects, with the terrain, specified in the form of a polygonal mesh. To solve this problem, algorithms are proposed that are based on the application of the separating axis theorem, geometric tests, as well as methods for segments and polygons clipping. In these algorithms, the computation of contact information about the intersection is realized by the method of delayed processing of triangles with sorting of penetration depths. This approach allows to effectively determine those triangles of the polygonal mesh that construct the contact patch boundary of the box with the terrain. The developed algorithms were tested in the software package of the virtual environment using the example of the interaction of virtual objects in the form of a box with various types of terrain.

Keywords: collision detection, bounding parallelepiped (box), terrain, polygonal mesh, contact manifold, separating axis theorem, clipping, virtual environment system

Литература

1. C. Ericson. Real-time collision detection. CRC Press, 2004.

2. G. van den Bergen. Collision detection in interactive 3D environments. Morgan Kaufmann Publishers, San Francisco, 2004.

3. Bounding volume. https://en.wikipedia.org/wiki/Bounding_volume (дата обращения: 18.03.2021).

4. J.A. Newth. Minkowski portal refinement and speculative contacts in Box2D. Master's Projects, 2013.

5. L. Olvang. Real-time collision detection with implicit objects. Department of Information Technology, Uppsala University, Sweden, 2010.

6. G. van den Bergen. A Fast and robust GJK Implementation for collision detection of convex objects, «Journal of Graphics Tools», vol. 4 (1999), no. 2, 7-25.

7. А.В. Санников. Численное моделирование динамики систем твердых деформируемых и жестких тел. Диссертация на соискание ученой степени кандидата физ.-мат. наук, Москва, 2015.

8. D. Gregorius. The separating axis test. «Game Developers Conference», 2013.

9. G. van den Bergen, and D. Gregorius. Game physics pearls. AK Peters/CRC Press, 2010.

10. T. Möller. A fast triangle-triangle intersection test, «Journal of Graphics Tools», vol. 2 (1997), no. 2, 25–30.

11. O. Tropp, A. Tal, I. Shimshoni. A fast triangle to triangle intersection test for collision detection: Research articles, «Computer Animation and Virtual Worlds», vol. 17 (2006), no. 5, 527–535.

12. D. Eberly. Dynamic collision detection using oriented bounding boxes. Technical report, Geometric Tools, Inc., 2002.

13. P. Terdiman. Contact generation for meshes. http://www.codercorner.com/MeshContacts.pdf (дата обращения: 18.03.2021).

14. T. Möller, and B. Trumbore. Fast, minimum storage ray-triangle intersection, «In Journal of graphics tools», V. 2 (1997), No. 1, 21–28.

15. Е.В. Страшнов, Л.А. Финагин. Определение коллизий аппроксимирующих сфер с рельефом местности. «Труды НИИСИ РАН», Т. 9 (2019), № 5, 111-118.

16. M. Cyrus, and J. Beck. Generalized two- and three-dimensional clipping, «Computers & Graphics», 1978, 23–28.

17. Е.В. Страшнов, М.А. Торгашев, А.В. Мальцев. Определение коллизий аппроксимирующих параллелепипедов и цилиндров. «Труды НИИСИ РАН», Т. 10 (2020), № 5-6, 139-146.

18. Е.В. Страшнов, М.А. Торгашев. Алгоритмы определения коллизий аппроксимирующих цилиндров с моделью рельефа местности. «International Journal of Open Information Technologies», Т. 8 (2020), № 6, 40-49.

19. М.В. Михайлюк, А.В. Мальцев, П.Ю. Тимохин, Е.В. Страшнов, Б.И. Крючков, В.М. Усов. Система виртуального окружения Virsim для имитационно-тренажерных комплексов подготовки космонавтов. «Пилотируемые полеты в космос», Т. 37 (2020), № 4, 72-95.

Планирование вычислений в системе реального времени с циклическим поступлением заданий

М.Г. Фуругян[1]

[1]ФИЦ ИУ РАН, Москва, Россия, rtsccas@yandex.ru

Аннотация. Рассматривается задача планирования вычислений в многопроцессорной системе реального времени с циклическим поступлением запросов на обработку поступающей информации. Данная задача формализована в виде задачи составления допустимого расписания выполнения комплекса работ в системе, состоящей из одного или нескольких идентичных процессоров. Работы характеризуются длительностями выполнения и директивными интервалами и допускают прерывания и переключения с одного процессора на другой. На множестве работ задано отношение частичного порядка. Разработан эвристический полиномиальный алгоритм поиска допустимого расписания, основанный на применении процедуры коррекции директивный интервалов и обобщении на многопроцессорный случай известного однопроцессорного алгоритма "относительной срочности"

Ключевые слова: многопроцессорная система, допустимое расписание с прерываниями, коррекция директивных интервалов, алгоритм "относительной срочности"

1. Введение

Задачи обработки периодически поступающей информации в системе реального времени возникают в самых разных областях деятельности человека, например, при разработке, испытаниях и эксплуатации сложных технических изделий. Так, при испытаниях самолетов используется вычислительная система жесткого реального времени, в основе которой лежит блок планирования вычислений. В эту систему с борта самолета в режиме реального времени с определенной частотой поступают данные, получаемые в ходе летного эксперимента. Эти данные должны быть обработаны при жестких временных ограничениях с тем, чтобы в нужное время передать на борт самолета указания о соответствующей корректировке проводимого испытания. Обработка заключается в выполнении комплекса программных модулей, которые могут обмениваться данными между собой. Эти обмены определяют отношения частичного порядка на множестве модулей.

В предлагаемом нами методе сначала выполняется процедура коррекции директивных интервалов на основе анализа графа частичного порядка. После выполнения указанной процедуры в однопроцессорном случае применяется RU-алгоритм, а для многопроцессорного случая разработана его модификация − многопроцессорный RU-алгоритм. В первом случае алгоритм работает так же корректно, как и при отсутствии отношения частичного порядка, а во втором случае алгоритм иногда выдает неверный ответ (отсутствие допустимого расписания в случае, когда оно существует), но, как показали многочисленные вычислительные эксперименты, число таких случаев невелико (не более 3% от числа проведенных экспериментов).

2. Постановка задачи

В многопроцессорной системе реального времени, состоящей из m идентичных процессоров, выполняются программные модули $M_1, M_2, ...M_K$. Требования на выполнение модуля M_i поступают циклически с целочисленным временным периодом p_i, $i = \overline{1, n}$, начиная с момента времени $t = 0$. Пусть P − наименьшее общее кратное величин $p_1, p_2, ..., p_K$ и пусть $k_i = P / p_i$. Таким образом, на временном интервале $[0; P]$ модуль M_i выполняется k_i раз и пусть M_{il_i} − его l_i-я активация. Модули M_{il_i}, $l_i = \overline{1, k_i}$, $i = \overline{1, K}$, могут обмениваться между собой параметрами. Кроме того, модуль M_{il_i} может начать выполняться только после завершения модуля M_{il_i-1}, $l_i = \overline{2, k_i}$, $i = \overline{1, K}$. Это определяет частичный порядок выполнения модулей M_{il_i}, $l_i = \overline{1, k_i}$, $i = \overline{1, K}$. Требуется составить такое расписание выполнения вычислений, чтобы успевать обработать очередной запрос на выполнение модуля M_i, $i = \overline{1, K}$. до поступления следующего запроса на этот модуль, или

установить, что такого расписания не существует. Таким образом, модуль M_{il_i} должен быть выполнен в интервале времени $[p_i(l_i-1);\ p_i l_i]$.

Данная задача может быть формализована и сформулирована следующим образом. Требуется выполнить комплекс работ (заданий) $W=\{w_1, w_2, ..., w_n\}$ с использованием m идентичных процессоров. Все работы допускают прерывания и переключения с одного процессора на другой, которые по предположению выполняются мгновенно, т.е. не требуют временных затрат. Не допускается параллельное выполнение одной работы несколькими процессорами, а также одновременное выполнение нескольких работ одним процессором. Каждая работа $w_i \in W$, $i=\overline{1,n}$, имеет следующие характеристики; длительность выполнения t_i и директивный интервал $[b_i,\ f_i]$ (т.е. работа w_i может быть начата не раньше момента времени b_i и должна быть завершена не позднее момента времени f_i). На множестве работ W задано отношение частичного порядка выполнения в виде ориентированного графа без циклов $G=(W, A)$, где W – множество узлов, A – множество дуг. Если $(w_i, w_j)\in A$, то работа w_j может быть начата только после завершения работы w_i .).

Расписание выполнения работ W будем задавать в виде вектор-функции $R(t)=(R_1(t), R_2(t), ..., R_n(t))$, где $R_i(t)=j$, $1\le j\le m$, если в момент времени t работа w_i выполняется процессором j; $R_i(t)=0$, если в момент времени t работа w_i не выполняется. При этом если $R_i(t)\ne 0$, $R_k(t)\ne 0$ и $i\ne k$, то $R_i(t)\ne R_k(t)$ при всех t, т.е. исключается одновременное выполнение двух или нескольких работ одним процессором.

Расписание называется допустимым, если каждой работы $w_i \in W$ суммарная длина интервалов, на которых $R_i(t)\ne 0$, равна t_i, и $R_i(t)=0$ при всех $t\notin[b_i,\ f_i]$, $i=\overline{1,n}$, т.е. если каждая работа полностью выполняется в своем директивном интервале.

Требуется определить, существует ли допустимое расписание выполнения заданий W, и построить его в случае положительного ответа.

Отметим, что в случае отсутствия отношений частичного порядка на множестве работ W задача может быть решена полиномиальным алгоритмом В.С. Танаева [2], основанным на сведении ее к задаче о максимальном потоке в сети специального вида. В случае, когда отношения предшествования заданы, задача является *NP*-трудной [1], даже если все работы имеют общий директивный интервал, и точного полиномиального алгоритма в этом случае не известно.

В настоящей статье предлагается эвристический полиномиальный алгоритм, являющийся обобщением однопроцессорного RU-алгоритма, применимого для случая без отношений предшествования [4]. В разд. 3 дается краткое описание этого алгоритма, а также приводится пример некорректной работы многопроцессорного RU-алгоритма. В разд. 4 приводится процедура коррекции директивных интервалов для задачи с отношениями предшествования, после которой может быть применен многопроцессорный RU-алгоритм, разработанный в настоящей статье (разд. 5, 6).

3. Однопроцессорный RU-алгоритм для задачи без отношений предшествования

В этом разделе дается описание однопроцессорного RU-алгоритма Э.Г. Коффмана [4].

Будем рассматривать однопроцессорную систему ($m=1$) в случае, когда отсутствуют ограничения на порядок выполнения работ ($A=\varnothing$). Пусть $W(t)$ – это множество активных (готовых к выполнению, но не завершенных) работ, т.е. $W(t)=\{w_i : b_i\le t,\ \textit{работа}\ w_i\ \textit{не завершена}\}$. Пусть, далее, $f_{i_0}=\min\limits_{i:w_i\in W(t)} f_i$, т.е. w_{i_0} – это активная работа с минимальным директивным сроком. Согласно RU-алгоритма работа w_{i_0} выполняется с момента времени t до тех пор, пока она не будет завершена, либо пока в некоторый момент времени $t_1>t$ не появится другая активная работа w_{i_1} с меньшим директивным сроком ($f_{i_1}<f_{i_0}$). В первом случае из активных работ выбирается работа с минимальным директивным сроком, которой выделяется процессор. Во втором случае процессор выделяется работе w_{i_1}, а работа w_{i_0} прерывается и переводится в разряд активных работ. Указанные действия повторяются до тех пор, пока не будет выполнен весь комплекс работ.

Из описания RU-алгоритма следует, что выбор активной работы с минимальным директивным сроком выполняется только либо в момент завершения одной из работ, либо в один из моментов времени b_i, $i=\overline{1,n}$, т.е. не более $2n$ раз.

Таким образом, вычислительная сложность однопроцессорного RU-алгоритма составляет $O(n^2)$. Если величины f_i, $i=\overline{1,n}$, хранить в виде кучи (пирамиды) с минимальным элементом в вершине [5], то вычислительная сложность алгоритма будет составлять $O(n\log n)$, поскольку построение кучи требует $O(n\log n)$ операций, а выбор элемента из вершины кучи и ее перестройка требуют $O(\log n)$ операций. Таким образом, однопроцессорный RU-алгоритм является полиномиальным. В [4] содержится доказательство корректности этого алгоритма. Иными словами, показано, что если в результате работы алгоритма построенное расписание не будет допустимым (одна или несколько работ завершаются позже своего директивного срока), то в этом случае допустимого расписания не существует.

4. Многопроцессорный RU-алгоритм для задачи без отношений предшествования

В настоящем разделе будет дана модификация однопроцессорного RU-алгоритма, применимая для многопроцессорной системы (многопроцессорный RU-алгоритм).

Предполагается, что теперь число процессоров m произвольное, а частичный порядок выполнения работ W, как и в разд. 3, отсутствует ($A=\varnothing$). Алгоритм заключается в том, что на свободные процессоры назначаются активные работы с наименьшими директивными сроками. Если свободных процессоров нет, то активная работа с меньшим директивным сроком назначается на процессор, на котором выполняется работа с наибольшим директивным сроком. При этом последняя прерывается и переводится в число активных работ. Такое назначение (замещение) может производится только по завершении одной из работ, либо в один из моментов b_i, $i=\overline{1,n}$, когда появляется новая активная работа.

Отметим, что из описания многопроцессорного RU-алгоритма следует, что выбор активной работы с минимальным директивным сроком и выбор выполняемой работы с максимальным директивным сроком производится не более $2n$ раз. Таким образом, вычислительная сложность предложенного алгоритма составляет $O(n^2m)$. Если хранить в виде двух куч директивные сроки выполняемых работ с максимальным элементом в вершине и директивные сроки активных работ с минимальным элементом в вершине, то вычислительная сложность алгоритма составит $O(n(\log n)\log m)$. Таким образом, предложенный алгоритм является полиномиальным.

Отметим, что в отличие от однопроцессорного случая многопроцессорный RU-алгоритм (при $m\ge 2$) не всегда работает корректно. Это видно из следующего примера. Пусть $m=2,\ n=3,\ b_i=0, i=1,2,3,\ f_1=\ =f_2=2,$ $f_3=3,\ t_1=t_2=1,\ t_3=3$. В этом случае в соответствии с описанием многопроцессорного RU-алгоритма работы w_1 и w_2 будут назначены на первый и второй процессоры соответственно, на которых они будут выполняться до их завершения. После чего работа w_3 будет назначена на первый процессор, на котором она будет выполняться до ее завершения в момент времени $t=4$.Таким образом, многопроцессорный RU-алгоритм допустимого расписания не построит. Однако, допустимое расписание существует и выглядит следующим образом: $R_1(t)=1$ при $t\in[0;1]$, $R_1(t)=0$ при $t\in[1;3]$, $R_2(t)=1$ при $t\in[1;2]$, $R_2(t)=0$ при $t\in[0;1)\cup(2;3]$, $R_3(t)=1$ при $t\in[0;3]$

Таким образом, на данном примере многопроцессорный RU-алгоритм сработал некорректно. Однако, как показали многочисленные машинные эксперименты, число таких случаев не более 3 % от общего числа испытаний [6].

5. Коррекция директивных интервалов в задаче с отношениями предшествования

Теперь рассмотрим задачу, сформулированную в разд. 2, т.е. когда число процессоров m произвольное и на множестве работ задано отношение частичного порядка в виде ориентированного графа без циклов $G(W, A)$.

Сначала разобьем множество вершин W графа G на непересекающиеся слои $X_1, X_2, ...,$ X_q ($X_i\cap X_j=\varnothing$ при $i\ne j$, $\bigcup_{j=1}^{q} X_j=W$) следующим образом. Пусть $P(w_i)$ – это множество всех непосредственных предшественников узла w_i, $i=\overline{1,n}$, в графе G, т.е. $P(w_i)=\{w_j\in W:(w_j, w_i)\in A\}$. Множество X_1 содержит все узлы из W, не имеющие непосредственных предшественников, т.е. $X_1=\{w\in W:P(w)=\varnothing\}$. Далее, пусть определено множество X_l, $1\le l<q$. Определим X_{l+1} как множество узлов, все непосредственные

предшественники которых содержатся в $\bigcup_{j=1}^{l} X_j$, т.е. $X_{l+1} = \{w \in W : P(w) \subseteq \bigcup_{j=1}^{l} X_j\}$.

Теперь проведем коррекцию моментов готовности b_i, $i = \overline{1, n}$, работ W последовательно для множеств $X_1, X_2, ..., X_q$ следующим образом:

$b_i^0 = b_i$ для $w_i \in X_1$;

$b_i^0 = \max_{j: w_j \in P(w_i)} \max(b_i, b_j + t_j)$ для $w_i \in X_j$, $j = \overline{2, q}$.

Поскольку при коррекции моментов готовности работ для каждого узла графа G выполняется фиксированное число операций для всех его предшественников, то вычислительная сложность этой процедуры составляет $O(n^2)$.

Рассмотрим, далее, другое разбиение множества узлов W на непересекающиеся слои $Y_1, Y_2, ..., Y_q$, ($Y_i \cap Y_j = \varnothing$ при $i \neq j$, $\bigcup_{j=1}^{q} Y_j = W$), построенное следующим образом

Пусть $Q(w_i)$ – это множество всех непосредственных последователей узла w_i, $i = \overline{1, n}$, в графе G, т.е. $Q(w_i) = \{w_j \in W : (w_i, w_j) \in A\}$. Множество Y_1 содержит все узлы из W, не имеющие непосредственных последователей, т.е. $Y_1 = \{w \in W : Q(w) = \varnothing\}$. Далее, пусть определено множество Y_l, $1 \le l < q$. Определим Y_{l+1} как множество узлов, все непосредственные последователи которых содержатся в $\bigcup_{j=1}^{l} Y_j$, т.е. $Y_{l+1} = \{w \in W : Q(w) \subseteq \bigcup_{j=1}^{l} Y_j\}$.

Теперь проведем коррекцию директивных сроков f_i, $i = \overline{1, n}$, работ W последовательно для множеств $Y_1, Y_2, ..., Y_q$ следующим образом:

$f_i^0 = f_i$ для $w_i \in Y_1$;

$f_i^0 = \min_{j: w_j \in Q(w_i)} \min(f_i, f_j - t_j)$ для $w_i \in Y_j$, $j = \overline{2, q}$.

Отметим, что число слоев X_j и Y_j совпадает, поскольку оно равно числу вершин в максимальном пути графа G.

Вычислительная сложность описанной процедуры составляет $O(n^2)$

6. Модифицированный RU-алгоритм в задаче с отношениями предшествования

В настоящем разделе представлена модификация RU-алгоритма для однопроцессорного и многопроцессорного случаев в задаче с отношениями предшествования на множестве работ. Этот алгоритм заключается в том, что сначала выполняется процедура коррекции директивных интервалов, как описано в разд. 4, а затем в однопроцессорном случае используется однопроцессорный RU-алгоритм, а в многопроцессорном случае − многопроцессорный RU-алгоритм.

Откорректированные директивные интервалы $[b_i^0, f_i^0]$, $i = \overline{1, n}$, обладают таким свойством, что если $(w_i, w_j) \in A$ (т.е. началу работы w_j должно предшествовать завершение работы w_i), то $b_i^0 \le b_j^0$ и $f_i^0 \le f_j^0$. Поэтому, выполнив предварительно процедуру коррекции директивных интервалов, далее в однопроцессорном случае можно использовать однопроцессорный RU-алгоритм, который всегда будет работать корректно. Применение многопроцессорного алгоритма не всегда приводит к правильному ответу, однако, как было отмечено в разд.4, процент числа таких случаев невелик

Вычислительная сложность модифицированного RU-алгоритма с учетом процедуры коррекции директивных интервалов составляет $O(n^2)$ в однопроцессорном случае и $O(n^2 m)$ в многопроцессорном случае

7. Заключение

Задача планирования вычислений в многопроцессорной системе реального времени с циклическим поступлением запросов на обработку поступающей информации формализована в виде задачи составления допустимого многопроцессорного расписания для комплекса работ с отношением частичного порядка. Рассмотрены однопроцессорный и многопроцессорный случаи. Разработан эвристический полиномиальный алгоритм поиска допустимого расписания, основанный на обобщении однопроцессорного алгоритма “относительной срочности” на многопроцессорный случай и применении метода коррекции директивных интервалов.

Planning Calculations in Real-Time System with Cyclical Receipt of Jobs

Meran Furugyan

Abstract. The problem of scheduling calculations in a real-time multiprocessor system with cyclic receipt of requests for processing of incoming information is discussed. This problem is formalized as the task of scheduling a set of jobs in a system consisting of one or more identical processors. Jobs are characterized by lengths of execution and directive intervals and allow interruptions and switching from one processor to another. A partial order relation is specified on the set of jobs. A heuristic polynomial algorithm for finding an acceptable schedule has been developed, based on the generalization of the known single-processor algorithm of "relative urgency" to the multiprocessor case and the use of the directive interval correction procedure.

Keywords: multiprocessor system, acceptable preemptive schedule, correction of directive intervals, "relative urgency" algorithm

Литература

1. М. Гэри, Д. Джонсон. Вычислительные машины и трудно решаемые задачи. М., Мир,1982.

2. В.С. Танаев, В.С. Гордон, Я.М. Шафранский. Теория расписаний. Одностадийные системы. М., Наука, 1984.

3. P. Brucker. Scheduling Algorithms. Heidelberg: Springer, 2007.

4. Э.Г. Коффман. Теория расписаний и вычислительные машины. М., Наука, 1984.

5. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы. Построение и анализ. (Второе издание). М., Вильямс, 2005.

6. М.Г. Фуругян. Некоторые алгоритмы анализа и синтеза многопроцессорных вычислительных систем реального времени // Программирование. 2014. № 1. С. 36 − 44.

Математический образ мышления или еще раз про электронную цифровую подпись

А.Б. Бетелин[1], А.А. Прилипко[2], Г.А. Прилипко[3], С.Г. Романюк[4], Д.В. Самборский[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ab@niisi.msk.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aaprilipko@niisi.msk.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, prilipko@niisi.msk.ru;

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

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, dsambor@niisi.msk.ru.

Аннотация. В XIX веке, при переходе России от феодализма к капитализму, когда массы крестьян превратились в наемных работников, потребовалась всеобщая грамотность: ведь чтобы расписываться в зарплатной ведомости, необходимо было научиться читать и писать. В настоящее время, при массовом распространении так называемых "цифровых технологий" (или "цифровизации общества") требуется массовая компьютерная грамотность. В статье разъясняется смысл термина "электронная подпись", в котором он используется в основополагающей работе [1], и на примере этого термина демонстрируются семантические проблемы, порождаемые отсутствием компьютерной грамотности в массе пользователей компьютеров и Интернета.

Ключевые слова: цифровая подпись, электронная подпись, электронная почта, криптография, криптограмма, ключ шифрования, асимметричное шифрование, цифровой, виртуальный, авторство, неотказуемость.

"Математику уже затем учить следует, что она ум в порядок приводит"
М.В.Ломоносов

1. Введение

Выяснение вопроса о том, что такое электронная подпись, разумно начать с вопроса, а что же такое обычная (рукописная) подпись. Зачем она нужна? О чем она свидетельствует или что доказывает?

Подпись может свидетельствовать о разном. Она может быть согласующей, утверждающей, ознакомительной, предписывающей и т.д., но ее универсальное свойство – свидетельство (доказательство) авторства.

Следующий очевидный вопрос: а возможны ли (существуют ли) другие способы доказательства авторства? Оказывается, существуют. В незапамятные времена для обмена сообщениями, смысл которых скрыт от всех, кроме адресата, была придумана криптография. Например, два человека могут обмениваться посланиями, которые зашифрованы ключом, известным только им двоим, и тогда никаких других доказательств авторства не требуется, авторство в этом случае очевидно и без подписи. (Вспомним, что шифрограммы не подписываются, а если и подписываются – то псевдонимом.) В традиционной криптографии и для зашифровывания, и для расшифровывания сообщения используется один и тот же ключ, который обеспечивает одновременно и секретность послания, и доказательство авторства.

2. Происхождение термина "электронная (цифровая) подпись"

Указанный факт оставался незамеченным до изобретения алгоритмов асимметричного шифрования [1]. При использовании этих алгоритмов предполагается, что каждому участнику обмена сообщениями выделена уникальная **пара** ключей шифрования: ключ **E**, называемый "открытым" (публичным, или общедоступным) и ключ **D**, называемый "закрытым" (личным, или секретным). Открытый ключ можно распространять без каких-либо ограничений, он считается известным кому угодно. Закрытый ключ должен держаться в секрете, он считается известным только одному человеку, владельцу этого ключа. Пара ключей подбирается таким образом, чтобы криптограмма S, полученная в результате применения любого из них к сообщению M, могла бы быть расшифрована только применением другого ключа из этой пары:

$$S = \mathbf{D}(M); \mathbf{E}(S) = \mathbf{E}(\mathbf{D}(M)) = M$$
$$S = \mathbf{E}(M); \mathbf{D}(S) = \mathbf{D}(\mathbf{E}(M)) = M$$

Иными словами, для зашифровывания сообщения можно использовать любой из двух ключей этой пары, но тогда для расшифровывания придется использовать другой. Главная идея алгоритма асимметричного шифрования заключается в двукратном зашифровывании сообщения M: сначала – применением секретного ключа отправителя, а затем – применением к полученной криптограмме S открытого ключа получателя. Расшифровывание конечной криптограммы выполняется в обратном порядке: сначала – секретным ключом получателя, а затем – открытым ключом отправителя. Первое зашифровывание обеспечивает доказательство авторства, а второе – секретность послания. Формально это можно записать следующим образом. Предположим, что участник переписки A желает отправить секретное сообщение участнику B.

Da – секретный ключ отправителя
Db – секретный ключ получателя
Ea – открытый ключ отправителя
Eb – открытый ключ получателя

Зашифровывание:

S = **D**a (M)
C = **E**b (S) = **E**b (**D**a (M))

Расшифровывание:

Db (C) = **D**b (**E**b (S)) = S
Ea (S) = **E**a (**D**a (M)) = M

Криптограмма S, полученная применением закрытого ключа **D** к сообщению M, используется для доказательства авторства: поскольку **D** – ключ секретный, никто кроме его владельца не может сформировать именно эту криптограмму для сообщения M. Отметим, что доказательством служит не наличие этой криптограммы самой по себе, а факт успешного применения к ней алгоритма дешифрования с использованием открытого ключа **E**a.

К моменту публикации статьи [1] стали привычными программы электронной почты и термин “электронное письмо”, поэтому промежуточную криптограмму S назвали “подписанным сообщением”, “подписью” и “электронной (цифровой) подписью”, а саму процедуру первого шифрования – “подписыванием сообщения”. Нужно подчеркнуть, что авторы статьи [1] понимали этот термин не буквально, а в переносном смысле, как метафору (недаром слово “подпись” было взято в кавычки). Дело в том, что никакого цифрового объекта, который можно было бы назвать “электронной подписью”, не существует. Принципиальное отличие ее от обычной подписи в том, что последняя воспринимается человеческим зрительным аппаратом непосредственно, и чтобы убедиться в ее наличии, не нужен ни компьютер с устройствами ввода/вывода, ни программы дешифрования (компьютер или устройства могут быть неисправны, программы – устаревшей версии и т.д.).

3. О математическом образе мышления

С XIX века науки делятся на естественные и гуманитарные. Имеется следующий критерий для выяснения, к какой из этих категорий относится наука. Любая наука имеет предмет исследования. Представим себе, что человек исчез. Сохранился ли (или исчез) предмет исследования? Если не исчез – наука естественная, если исчез – гуманитарная. Согласно этому критерию, астрономия – наука естественная (ведь звезды не исчезнут от того, что их некому будет наблюдать), а медицина (наука о болезнях и здоровье человека) – наука гуманитарная. Предмет исследования математики – абстракции, идеализированные сущности, существующие только в головах математиков, поэтому математика (с точки зрения этого критерия) – наука гуманитарная. Этот вывод противоречит общепринятому представлению, согласно которому математика – наука естественная, не говоря уже о “массовом сознании”, в котором, например, математика и физика – это вообще одна и та же наука.

Особенность математического образа мышления в том, что математик считает изучаемые им объекты “объективной реальностью”, существующей и вне его головы, поэтому при исчезновении всех людей объекты исследования математиков не исчезнут. Такое представление не лишено оснований. Если математические абстракции существуют только в головах математиков, то почему они одни и те же во всех головах? Почему идеи неэвклидовой геометрии пришли в голову практически одновременно трем незнакомым друг с другом математикам, жившим в разных странах и говорящим на разных языках? В этом факте есть что-то “от Бога”, считают математики.

Другая особенность математического образа мышления – в интерпретации слова “факт”. Это слово имеет разный смысл, например, для физиков и математиков. В середине XX века в период становления квантовой механики один из советских физиков произнес следующие слова: ”Современному физику порой кажется, что почва уходит из-под ног. Это ощущение обманчиво, потому что почва эта – факты.” Для физика (и

других нематематиков) факт – это нечто, допускающее экспериментальную проверку. Утверждение “Сумма углов треугольника (на плоскости) равна 180 градусов” для физика является логическим следствием, выводом из аксиом геометрии, но не фактом: измерение угла всегда выполняется с некоторой погрешностью. Но математики называют такие утверждения о математических объектах фактами. (Кстати, утверждение “Сумма углов любого треугольника (на плоскости) равна 180 градусов” не является для физика фактом просто потому, что экспериментально невозможно измерить углы всех треугольников на свете.)

Еще одно свойство математического образа мышления – невероятно широкое использование математиками известных слов языка для обозначения придумываемых ими объектов. [2] Одно из негативных последствий этого – создание (в головах нематематиков) ложной совокупности ассоциаций, что произошло с термином “электронная цифровая подпись”. Для обычного человека (нематематика) эти слова ассоциируется прежде всего с обычной подписью, иероглифом (или чем-то таким, что можно увидеть или изобразить), несмотря на прилагательные “электронная”, “цифровая” и кавычки.

4. О виртуальном

В английском языке есть слово ***virtual***, имеющее два противоположных по смыслу значения:

(1) гипотетический, воображаемый, мнимый и

(2) фактический, действительный.

Слово это не имеет эквивалента в русском языке и переводится как ***виртуальный***. В физике давно укоренился термин “виртуальная частица”, а в вычислительной технике и информатике это прилагательное используется чрезвычайно широко: виртуальная машина, память; виртуальный терминал, диск; виртуальное устройство, соединение и т.д. Это слово как нельзя лучше подходит для обозначения того, что называют электронной цифровой подписью (ЭЦП): ЭЦП как цифрового объекта не существует, т.е. она *мнимая* (ей не соответствует даже двоичного изображения), и в то же время она дает *фактический* эффект, доказательство авторства. Однако авторы основополагающей статьи [1] предпочли слово ***digital***, посчитав ***virtual*** слишком перегруженным. В результате этого выбора смысловой акцент (в сознании нематематиков) сместился со слова “цифровая” на слово “подпись”, чего (возможно) не произошло бы, если бы подпись назвали “виртуальной”.

5. Последствия терминологического выбора

Дело в том, что в массовом сознании слово “виртуальный” ассоциируется с чем-то ненастоящим, даже фальшивым, а слова “электронный” и “цифровой” – с чем-то высоконаучным, поэтому считается, то ЭЦП – это примерно то же самое, что и обычная подпись, и что они эквивалентны, а юристы даже утверждают, что обычная подпись и электронная “имеют одинаковую юридическую силу”.

Согласно определению, извлеченному из интернета, “Электронная подпись – это атрибут цифрового документа, полученный в результате криптографического преобразования ...”. Можно догадаться, что речь идет о криптограмме S цифрового документа M. На первый взгляд – все выглядит логично: если обычная подпись – атрибут обычного документа, то цифровая подпись должна быть атрибутом цифрового документа. Однако это поверхностное сравнение порождает много недоразумений.

(1) Криптограмма документа – это другой документ, а не его атрибут (никто ведь не считает атрибутом документа, например, его копию). На это математик может возразить, что он может все что угодно назвать чем угодно, такова специфика его работы. Однако он может не учесть, что при этом в его мозгу происходит перестройка графа ассоциаций, которая требует времени (вот почему “понимание” – процесс длительный), а эта деятельность, привычная для математика, часто оказывается непосильной для нематематиков и вызывает у них ненужные вопросы вроде следующих.

(2) Обычная подпись – одна и та же на всех документах, а электронная? Из определения следует, то она для каждого документа – своя, а значит, она не совсем то же самое, что рукописная.

(3) А если у документа несколько подписей, то сколько существует атрибутов “электронная подпись” и в каком порядке их нужно вычислять?

Подобные вопросы не возникают, если с самого начала осознавать, что обычная подпись и электронная – совершенно разные объекты, и ***единственное***, что у них общего, заключается в том, что и та, и другая служат доказательством авторства.

Еще одно последствие терминологического выбора – расширенное толкование термина “электронная подпись”. Например, в некоторых Интернет-публикациях электронной подписью называют пару Login/пароль. Другой пример – из банковской деятельности. В докомпьютерные времена при приеме платежа от физического лица в платежных документах, выписываемых

банком, фигурировала подпись вносителя. Теперь в результате цифровизации вместо слов “подпись вносителя” печатаются слова “подписано электронной подписью <Иванова И.И.>”, а подписывать документы вноситель больше не должен (“автоматизация”). В данном случае за этими словами скрывается некое дополнение к процедуре оформления платежа, состоящее в том, что работник банка просит клиента вставить банковскую карту в считыватель и ввести пин-код, а слова “подписано электронной подписью”, печатаемые в платежных документах, означают всего-навсего, что пин-код введен правильный. Никакой “электронной подписи” (в изначальном смысле) здесь нет, как нет и криптографии.

Но главное последствие терминологического выбора – дезориентация массового пользователя. Электронная подпись считается эквивалентной обычной (или “имеющей ту же юридическую силу”) на том основании, что она тоже доказывает авторство, однако при этом не акцентируется внимание на том, что это справедливо лишь при (практически непроверяемом) **условии сохранения ключа шифрования в тайне**. Вместо этого говорится о таких разновидностях (свойствах) электронной подписи, как “простая” и “усиленная”, а также “квалифицированная” и “неквалифицированная”. При ближайшем рассмотрении выясняется, что эти прилагательные относятся к процедуре аутентификации пользователя, которому выделяются ключи шифрования, именно эта процедура может быть “простой”, “усиленной” и т.д.

Сложившаяся терминология похожа на “собачий язык” из воспоминаний Н.В.Тимофеева-Ресовского [3], на котором в секретных отчетах писали “калий” или “натрий”, а подразумевали – “уран”. Вот несколько примеров выражений, относящихся к электронной подписи, с переводом на “человечий” язык.

Ключи электронной подписи – *ключи шифрования*

Подписанное сообщение – *шифрограмма S (результат применения закрытого ключа отправителя)*

Подписать сообщение – *зашифровать закрытым ключом отправителя*

Сертификат ключа электронной подписи – *открытый ключ отправителя*

Подписан сертификатом – *зашифрован закрытым ключом, соответствующим данному открытому ключу*

6. Вместо заключения

Необходимым элементом компьютерной грамотности является знание о том, что цифровые объекты (программы, файлы и др.) не воспринимаются органами чувств человека непосредственно, т.е. они невидимы, неслышимы и т.д., и что для них придуман специальный термин: “неосязаемости”. (Правда, говорят, что файл можно визуализировать, распечатав его на бумаге, однако то, что распечатано, есть не сам файл, а его копия, к тому же часто называемая “твердой”, hard copy.)

Еще одним элементом компьютерной грамотности является знание о том, что для цифрового объекта существует унифицированный образ в виде цепочки двоичных цифр. (Заметим, что **образ**, например, отражение человека в зеркале – это **не копия**.) Особенность цифровой подписи в мире цифровых объектов состоит в том, что для нее не существует даже двоичного образа: двоичный образ существует для ключей шифрования, криптограмм, данных и метаданных файла, но не для цифровой подписи. Это порождает вопрос, а как же удостовериться в наличии того, что невидимо и не имеет даже двоичного образа, и как следствие – стремление “материализовать” нематериальное. В результате применения к цифровой подписи тех же слов, что и к обычной, родился аналог “собачьего языка” [3], к которому легко привыкают математики, но не “массовый пользователь”. Осознание того, что цифровая подпись – объект виртуальный, помогает избежать семантических трудностей.

Единственное качество, объединяющее обычную и цифровую подпись – доказательство авторства (которое в некоторых публикациях называют “неотказуемостью”, подразумевая, видимо, “неотказуемость от авторства”). Доказательством авторства в случае электронной подписи служит не иероглиф, начертанный на бумаге, а логические рассуждения, которые на практике могут оказаться более длинными, чем говорилось выше. Например, зашифровываться может не само сообщение M, а его контрольная сумма (хэш-сумма), вычисляемая по некоторому стандартному алгоритму, и доказательством авторства в этом случае будет совпадение вычисленной контрольной суммы с расшифрованной.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Выполнение фундаментальных научных исследований ГП 47) по теме № 0580-2021-0007 «Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления»

Mathematical Way of Thinking or Once Again about Electronic Digital Signature

A.B. Betelin, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. In the nineteenth century, during Russia's transition from feudalism to capitalism, when the masses of peasants turned into wage-earners, universal literacy was required: after all, to sign the payroll, you had to learn to read and write. At present, with the mass spread of so-called “digital technologies” (or “digitalization of society”), mass computer literacy is required. The article explains the meaning of the term “electronic signature", in which it is used in the fundamental work [1], and the semantic problems generated by the lack of computer literacy in the mass of computer and Internet users are demonstrated by the example of this term.

Keywords: digital signature, electronic signature, email, cryptography, cryptogram, encryption key, asymmetric encryption, digital, virtual, authorship, non-repudiation.

Литература

1. R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. “ACM”, Vol. 21 (1978), Iss. 2, 120—126.

2. Успенский В.А. Математическое и гуманитарное: преодоление барьера. М., МЦНМО, 2012.

3. Тимофеев-Ресовский Н.В. Воспоминания. М., «Вагриус», 2008, с.331-335. https://vikent.ru/ /enc/2558 (дата обращения 16.03.2021).

Подписано в печать 23.04.2021 г.
Формат 60x90/8
Печать цифровая. Печатных листов 6,0
Тираж 100 экз. Заказ № 321

Отпечатано в ФГУП «Издательство «Наука»
(Типография «Наука»)
121099, Москва, Шубинский пер., 6