ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №4(31), 2016, с. 119–143

УДК 004.45

Ю. В. Шевчук, А. Ю. Шевчук

Виртуальная машина «Etherbox32vm»

Аннотация. Описана организация виртуальной машины Etherbox32vm, предназначенной для реализации сценариев функционирования узлов сенсорной сети: дисциплин опроса датчиков, управления исполнительными механизмами, первичной обработки данных, полученных от датчиков. Реализация логики поведения сенсорных узлов в виртуальной машине позволяет удаленно настраивать узлы уже развернутой гетерогенной сенсорной сети на выполнение новых функций. Виртуальная машина Etherbox32vm допускает реализацию на микроконтроллерах с малым объемом оперативной памяти.

Ключевые слова и фразы: сенсорные сети, виртуальная машина, стековая машина.

Введение

Мы рассматриваем сенсорную сеть, состоящую из управляющей станции и множества распределенных в пространстве программируемых сенсорных узлов, связанных разнородными каналами связи. Каждый сенсорный узел обслуживает некоторое количество датчиков и/или исполнительных механизмов. Работа сенсорного узла состоит в периодическом опросе датчиков, первичной обработке полученных данных и передаче данных на управляющую станцию, а также в управлении исполнительными механизмами по командам от управляющей станции.

И дисциплину опроса датчиков, и алгоритмы первичной обработки данных, и алгоритмы управления исполнительными механизмами невозможно жестко задать в программе сенсорного узла: они могут как отличаться от узла к узлу, так и меняться во времени. Таким образом, возникает проблема задания каждому узлу сценария поведения в соответствии с его назначением, с возможностью удаленного изменения сценариев.

Решать эту проблему можно разными способами. Например, если сенсорный узел реализован на базе миниатюрного компьютера под управлением ОС семейства UN*X (такой как [1] или [2]), можно было бы задавать сценарии в форме скриптов на shell или другом интерпретируемом языке. Или (диаметрально противоположный подход) можно загружать сценарии в форме программы, скомпилированной в код процессора сенсорного узла. Возможны и другие варианты. При выборе решения необходимо учитывать следующие обстоятельства:

- на данном этапе развития технологии среди сенсорных узлов преобладают устройства с ограниченными ресурсами, например на базе микроконтроллеров с объемом памяти порядка 30КБ, так что выбранное решение должно допускать компактную реализацию;
- сенсорный узел может быть устройством с автономным питанием, для которого экономия циклов процессора означает увеличение времени автономной работы. Поэтому выбранное решение не должно быть затратным с точки зрения ресурсов процессора;
- канал связи с сенсорным узлом может быть низкоскоростным (LoWPAN) или помегабайтно оплачиваемым (GSM). Поэтому как сценарии, так и пересылаемые ими на управляющую станцию данные должны быть компактными;
- мы рассматриваем гетерогенные сенсорные сети, в которых разные узлы имеют разную аппаратную реализацию и разное количество ресурсов. Желательно иметь единообразный механизм конфигурации сенсорных узлов, скрывающий различия аппаратной реализации, но при этом не отнимающий возможность использовать полностью ресурсы более мощных сенсорных узлов.

Предлагаемое в настоящей статье решение состоит в том, что сценарии представляются в виде байткода специализированной виртуальной машины Etherbox32vm. Виртуализация абстрагирует программу от аппаратуры сенсорного узла, что решает проблему гетерогенности сенсорных узлов. Виртуальная машина спроектирована в расчете на компактную реализацию самой виртуальной машины и компактность байт-кода. В байт-коде существует возможность расширения системы команд, что позволяет при необходимости эффективно реализовать операции, затратные с точки зрения ресурсов процессора. Возможность псевдопараллельного выполнения нескольких взаимодействующих процессов в виртуальной машине позволяет реализовать одновременно несколько сценариев на более мощных сенсорных узлах.

Содержание статьи организовано следующим образом. В разделе 1 приведен краткий обзор альтернативных решений проблемы удаленной реконфигурации сенсорных узлов. В разделе 2 дано общее описание виртуальной машины Etherbox32vm и рассматриваются ее отличительные черты.

1. Обзор возможных решений

Проблема удаленной реконфигурации сенсорных узлов не является новой. Существуют и применяются на практике самые разные способы ее решения.

1.1. Удаленное перепрограммирование сенсорных узлов

Суть подхода заключается в полной удаленной замене образа программного обеспечения обновляемого устройства.

Такой подход имеет ряд недостатков. Во-первых, при таком подходе любое обновление требует перезагрузки обновляемого устройства. Это может вызывать нежелательные перерывы в работе сенсорной сети, особенно заметные в случае массовых или частых обновлений.

Во-вторых, такой подход плохо согласуется с гетерогенным характером сенсорной сети: для сенсорных узлов с отличающимся аппаратным обеспечением необходимо создавать разные версии одного обновления.

В-третьих, каждое такое обновление содержит избыточную информацию, поскольку значительная часть программного обеспечения остается без изменений. Такое увеличение накладных расходов нежелательно при передаче массовых обновлений по каналам связи с малой пропускной способностью.

Наконец, обновление собственного кода микроконтроллера сенсорного узла является проблематичным с точки зрения безопасности: ошибка программиста, содержащаяся в обновлении, или ошибка в процессе обновления могут привести к полной неработоспособности устройства. Перед обновлением образа программного обеспечения удаленных узлов необходимо тестирование загружаемого кода на локальном узле — операция, требующая времени и, возможно, участия человека. Таким образом, этот механизм не годится для оперативного изменения сценариев работы узлов.

Существуют также способы частичной замены образа на основе отличий между текущим образом и обновленным, что уменьшает накладные расходы на передачу обновлений, однако создает еще большие трудности с гетерогенностью [3].

Стоит отметить, что возможность полной удаленной замены программного обеспечения сенсорных узлов безусловно полезна для устранения ошибок и добавления новых возможностей во встроенное программное обеспечение сенсорных узлов. Однако для задания сценариев работы сенсорных узлов она плохо подходит.

1.2. Динамическое подключение исполняемых модулей Contiki

В операционной системе Contiki [4] существует механизм динамической загрузки исполняемых программных модулей. Этот механизм позволяет снизить объем обновлений по сравнению с полной заменой образа, а также снимает необходимость перезагружать устройство после каждого обновления. Однако проблемы с надежностью и гетерогенностью сети остаются.

1.3. Виртуализация

Радикальным решением проблемы гетерогенности является частичная реализация программного обеспечения сенсорных узлов в виде байт-кода виртуальной машины, работающей на каждом узле.

Существуют реализации для маломощных устройств как виртуальных машин общего назначения, например JVM [3, 5, 6], так и специализированных виртуальных машин, спроектированных для сенсорных сетей, таких как VM* [5], Maté [7], CVM [3].

Представление части ПО сенсорного узла в кодах виртуальной машины дает ряд преимуществ по сравнению с обновлением собственного кода устройства:

- минимизация размера передаваемых по сети обновлений снижает энергопотребление устройства;
- защищенная среда исполнения повышает устойчивость устройства к ошибкам времени исполнения.

Основным недостатком виртуализации является низкая, по сравнению с собственным кодом процессора сенсорного узла, производительность и повышенное энергопотребление [3]. Частично этот недостаток можно устранить, реализовав требовательные к производительности части программ в виде отдельных инструкций виртуальной машины.

Как отмечено выше, описываемая в настоящей статье виртуальная машина Etherbox32vm не является единственной существующей в настоящее время виртуальной машиной для сенсорных сетей. Однако, она создавалась и развивалась независимо от остальных и содержит ряд оригинальных решений, которые могут оказаться интересны читателю.

2. Виртуальная машина Etherbox32vm

2.1. Место Etherbox32vm в программном обеспечении сенсорного узла

Программы в байт-коде Etherbox32vm поступают в сенсорный узел через транспортный интерфейс сенсорного узла. В гетерогенной сенсорной сети сетевой уровень реализован с использованием протокола интернет: IPv4 или IPv6; для доставки байт-кода узлу используется специализированный прикладной протокол, основанный на протоколе UDP [8]. Описание протокола выходит за рамки настоящей статьи; отметим только, что в одном пакете UDP содержится законченная программа для виртуальной машины. Максимальный размер UDP-пакета с учетом фрагментации составляет 64КБ. В имеющихся на сегодняшний день реализациях сенсорных узлов с Etherbox32vm фрагментация IP-пакетов не поддерживается, так что максимальный размер программы вместе с накладными расходами на протокол не может превышать MTU[1] сети: 1500 или 1280 байтов в разных реализациях. Благодаря компактности кода виртуальной машины Etherbox32vm, этого оказывается достаточно для возникающих на практике задач, а если размер кода все же превысит максимальный, есть возможность выноса части кода в описанные ниже в разделе 2.7 публичные функции.

[1]Maximum transmission unit — максимальный размер пакета сети второго уровня, доступный для использования протоколом IP

Минимальная программа для Etherbox32vm может состоять всего из одной команды. За небольшой размер мы будем называть программы для Etherbox32vm словом *проглет* (англ. proglet[2]).

После включения питания сенсорный узел подключается к транспортной сети и ждет UDP-пакетов с проглетами от управляющей станции. Проглеты могут быть разной сложности. В простейшем случае, проглет может содержать единственную команду для совершения узлом определенного действия (считывания информации с датчика или однократного воздействия на исполнительный механизм). В этом случае проглет отрабатывает команды, завершается и на управляющую станцию возвращается UDP-пакет, содержащий исходный проглет со всеми изменениями, которые произошли в нем в ходе выполнения проглета на виртуальной машине. Данные, полученные в результате работы проглета (показания датчиков, статусы периферийных устройств и т.п.) при этом оказываются записанными в теле проглета в специально отведенных участках программы (см. раздел 2.9). Таким образом, управляющая станция получает ответ от сенсорной сети в том же формате, в каком был сформулирован запрос.

Проглет также может быть и более сложной программой, реализующей сценарий поведения узла на длительном промежутке времени, что дает сенсорным узлам возможность принимать определенные решения с учетом полученных от датчиков данных самостоятельно, без вмешательства управляющей станции. Типичными функциями сценариев являются:

- задание дисциплины (порядка и частоты) опроса датчиков;
- первичная обработка полученных от датчиков данных: агрегирование, фильтрация, принятие решения об отправке данных на управляющую станцию;
- задание дисциплины (порядка, частоты, условий) активации исполнительных механизмов.

На одном устройстве может исполняться одновременно несколько проглетов, каждый из которых выполняется в собственном адресном пространстве, но может взаимодействовать с другими проглетами с помощью механизма публичных функций (см. раздел 2.7).

[2]Proglet — «очень маленькая программа», по аналогии с Piglet — «очень маленькое существо» [9].

Для того чтобы изменить поведение узла, управляющая станция передает сенсорному узлу новый проглет, который заменяет один из уже исполняемых узлом проглетов или начинает исполняться параллельно с ними. Замена проглетов (завершение старого по прибытии нового) также реализуется с помощью механизма публичных функций.

2.2. Архитектура Etherbox32vm

Etherbox32vm является стековой машиной. Вычислительный и программный стеки объединены в один стек.

По сравнению с регистровой архитектурой, стековая архитектура позволяет уменьшить объем кода, исполняемого виртуальной машиной за счет отсутствия полей выбора операндов в коде инструкций [10, 11]. В момент интерпретации большей части инструкций, операнды находятся уже на стеке и именно в том порядке, в каком их ожидает интерпретатор.

Стек виртуальной машины содержится полностью в оперативной памяти интерпретатора, вне адресного пространства проглета. Это упрощает реализацию и позволяет не делать дополнительных предположений об архитектуре физического процессора, на котором реализована виртуальная машина (в частности, о количестве регистров). Таким образом, одна реализация виртуальной машины может использоватсья на устройствах разной мощности для выполнения одного и того же кода, при этом на устройствах с большим объемом оперативной памяти появляется возможность выполнения функций с большим числом локальных переменных и вычисления более сложных выражений за счет увеличения объема стека.

Такая реализация стека является субоптимальной в плане производительности, однако в нашем случае этот фактор не имеет большого значения: реализация сценариев работы сенсорных узлов не предполагают существенной вычислительной нагрузки. В литературе описаны способы выделения регистров процессора и оптимизации порядка инструкций [12, 13] для стековых машин, которые можно использовать для увеличения производительности в случае возникновения такой необходимости.

Элементы стека являются целыми значениями длиной 32 бита. Куча (heap) в памяти проглета отсутствует, динамическое выделение

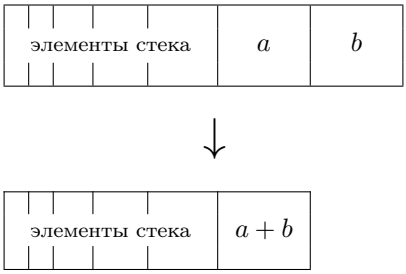

Рис. 1. Стек до и после команды add

памяти не предусмотрено. Адреса возврата из функций хранятся в отдельном *стеке вызовов* в памяти интрепретатора, вне адресного пространства проглета. Работа со стеком вызовов осуществляется только командами вызова функций и возврата из них; никакие другие манипуляции с содержимым стека вызовов невозможны.

Каждый проглет выполняется в собственном виртуальном адресном пространстве, которое инициализируется содержимым UDP-пакета. Выполнение начинается с первой инструкции проглета (с адреса 0).

2.3. Система команд Etherbox32vm

Большинство команд Etherbox32vm представлены в памяти одним байтом, содержащим код инструкции. Операнды команды, если они имеются, должны быть предварительно помещены на стек. Если команда в результате своей работы возвращает значение, оно также помещается на стек.

Таким образом работают, например, команды арифметических и логических операций, как схематически показано на рисунке 1.

Существуют также команды, операнды которых передаются в виде непосредственных значений, следующих за байтом кода команды. Такие команды могут быть представлены в памяти несколькими байтами. Порядок байтов в непосредственных значениях — little endian (первый байт содержит младшие биты, последний старшие).

Есть также несколько команд, у которых часть кода команды представляет собой непосредственный операнд или его часть. Например, команда push11 представлена двумя байтами. Эта команда загружает на стек значение, содержащееся в младших 11 битах команды: 8 бит

младшего байта и 3 младших бита старшего байта. Пять старших битов кода команды идентифицируют команду как `push11`. Подобная ей команда `push6` кодируется одним байтом и загружает на стек непосредственное значение длиной до 6 бит. Таким образом, загрузка часто встречающихся в программах малых констант экономично кодируется одним или двумя байтами, в то время как для кодирования команды `push32`, загружающей любое 32-битное значение, потребуется пять байтов.

Результатом работы некоторых команд является модификация массива байтов, следующих за кодом команды. К таким командам относятся, например, команды работы со внешними интерфейсами, описанные ниже в разделе 2.9.

Etherbox32vm реализует набор инструкций, достаточный для реализации Си-подобного языка. Проглеты, состоящие из нескольких команд, легко выражаются на языке ассемблера. Для более сложных проглетов существует компилятор Си-подобного языка Etherbox2 в байт-код Etherbox32vm [**14**].

2.4. Работа со стеком и памятью

В машине Etherbox32vm единственный стек выполняет функции вычислительного стека, используемого для аргументов и результатов команд, и программного стека, используемого для локальных переменных, аргументов и результатов функций. Поэтому помимо команд работы с вершиной стека, традиционных для этой структуры данных (добавление, удаление, дублирование элемента на вершине стека — `push, pop, dup`), предусмотрены дополнительные команды:

`dupn N` — копирование на верхушку стека значения, находящегося в стеке на глубине N;

`pushn VAL N` — модификация значения, находящегося в стеке на глубине N;

`popn N` — удаление N элементов с верхушки стека.

Дополнительные команды позволяют работать с находящимися на стеке локальными переменными и аргументами функций. Поскольку положение локальных переменных относительно начала стекового фрейма остается неизменным, для чтения и записи локальных данных функции оказывается достаточно двух команд (первая из которых — загрузка константы `N`), равно как и для удаления стекового фрейма

Таблица 1. Структура команды `data`

Поле	`opcode`	`length`	`data`
Длина *в байтах*	1	2	*length*

при возврате из функции (см. также раздел 2.6, «Работа с подпрограммами»).

Загрузка значений из памяти на стек осуществляется командами `load8, load16, load32`. Эти команды получают единственный аргумент на стеке: адрес загружаемого значения. Сохранение значения, находящегося на вершине стека, в память проглета, осуществляется командами `store8, store16, store32`.

Адресация памяти производится по абсолютному адресу в виртуальном адресном пространстве проглета.

2.5. Сегмент данных

Сегмент данных проглета содержится в теле команды data. Структура команды data описана в таблице 1. Традиционный для языка Си сегмент bss отсутствует, неинициализированные данные помещаются в сегмент данных и инициализируются нулями. Это упрощение плохо согласуется с заявленной целью снижения объема передаваемых по сети данных, но в перспективе планируется реализовать в протоколе сжатие данных, что нейтрализует этот негативный эффект.

Если статические данные присутствуют в программе, команда data записывается в последовательности команд первой. Таким образом, значения адресов статических данных проглета оказываются наименьшими, что увеличивает долю компактных команд `push8` среди команд загрузки на стек адресов переменных.

Загрузка значений из сегмента данных на стек и обратно производится командами `load` и `store`, соответственно.

2.6. Работа с подпрограммами

Вызов функций производится командой `call`. Команда `call` совершает переход по указанному адресу, предварительно сохранив адрес следующей инструкции в стек возвратов. Стек возвратов является частью контекста исполнения проглета и реализован отдельно от вычислительного стека.

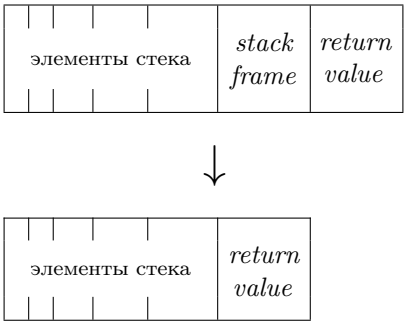

Рис. 2. Стек до и после команды `smartret`

Команда `ret` производит возврат из текущей функции, совершая переход по адресу, находящемуся на верхушке стека возвратов текущего проглета.

Уровень начала стекового фрейма текущей функции также хранится в стеке возвратов. Стековый фрейм, содержащий аргументы и локальные переменные функции, удаляется командой `popn` непосредственно перед возвратом из функции.

Однако в случае, если функция возвращает значение, при удалении стекового фрейма требуется сохранить находящееся в момент возврата на вершине стека возвращаемое значение. Для этого была введена специальная команда `smartret` (см. рис. 2).

2.7. Публичные функции

Взаимодействие между проглетами, выполняющимися на одном сенсорном узле, осуществляется с помощью механизма публичных функций. Каждый проглет может сделать несколько своих функций публичными, т.е. доступными для вызова другими проглетами. Взаимодействие с помощью публичных функций напоминает работу с объектами с помощью функций-методов в объектно-ориентированных языках: непосредственная манипуляция данными объекта извне невозможна, можно только вызывать предоставленные объектом методы. Аналогично, прямой доступ из одного проглета в адресное пространство другого невозможен, но возможен контролируемый доступ, как на чтение, так и на запись, путем вызова публичных функций.

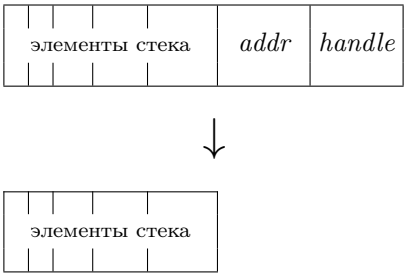

Рис. 3. Стек до и после команды `link`

Регистрация публичной функции осуществляется посредством команды `link`. Команда `link` получает два аргумента на стеке: адрес, по которому в текущем проглете находится функция, и числовой дескриптор, который требуется ей присвоить (рис. 3).

После того, как публичная функция зарегистрирована, она может быть вызвана из другого проглета, работающего на этом же устройстве, командой `xcall`. Команда `xcall` получает единственный аргумент: дескприптор вызываемой публичной функции.

Код вызываемой публичной функции выполняется в контексте текущего проглета (а значит, и с текущим стеком), но в адресном пространстве того проглета, в котором находится код публичной функции.

На практике это означает, что аргументы можно передавать в публичную функцию так же, как в обычную функцию, и результат публичной функции тоже возвращается на стеке, как из обычной функции. Однако передать в качестве аргумента указатель на значение в памяти текущего проглета невозможно: при разыменовании такого указателя будет получено значение из памяти проглета публичной функции.

При этом существует необходимость каким-то образом передавать в публичные функции массивы значений, а также возвращать массивы значений из публичных функций. Это достигается с помощью команд `copyargs` и `copyret`, предназначенных для копирования данных из памяти одного проглета в память другого. Обе команды получают свои аргументы на стеке.

Команда `copyargs` копирует данные из вызывающего проглета в текущий. В публичную функцию передается значение `pack`, со-

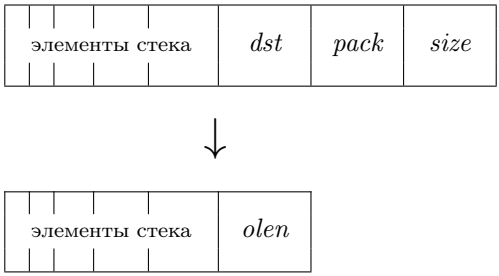

Рис. 4. Стек до и после команд `copyret` и `copyargs`

держащее в упакованном виде адрес и длину буфера, содержащего данные, а уже сама функция вызывает `copyargs`, копируя данные из вызывающего проглета в буфер в вызываемом проглете.

Команда `copyret` копирует данные из последнего вызывавшегося проглета в текущий. Чтобы возвратить массив, публичная функция возвращает значение `pack`, после чего вызывающая функция копирует данные из вызываемого проглета в буфер в вызывающем проглете командой `copyret`.

Действие команд `copyargs` и `copyret` на стек показано на рисунке 4), где `dst` и `size` — адрес и размер локального буфера; `pack` — адрес и длина удаленного буфера, содержащего копируемые данные, в упакованном виде: `pack = len<<16|src`; `olen` — число скопированных байтов.

2.7.1. *Публичные функции и выполнение проглета*

Вызов публичной функции никак не влияет на выполнение проглета, которому принадлежит функция. Возможно, такая функциональность будет добавлена позже, например в форме команды `wakeup`, которая при вызове внутри публичной функции возвращает проглет в очередь готовых к исполнению, если проглет исполнял команду задержки (например, `mdelay`). Это позволит приблизить семантику вызова публичной функции к семантике обмена сообщениями, в которой присутствует как передача инфрмациии, так и синхронизация.

Еще один режим использования публичных функций — в качестве библиотечных. В этом случае основной поток управления проглета после регистрации публичных функций командой `link` завершается,

но поскольку в теле проглета остались зарегистрированные публичные функции, занимаемая проглетом память не освобождается как обычно при завершении проглета, и функции остаются доступны для вызова. Память проглета будет освобождена, если регистрацию всех его публичных функций отменить командой `unlink`, или заменить командой `link` из других проглетов.

2.8. Ветвление и базовые блоки

При реализации компилятора или программировании на языке ассемблера для стековой машины возникает проблема согласования стека при передаче управления. Проблема связана с тем, что управление может попасть в один и тот же базовый блок[3] несколькими различными способами, так что в начале базового блока текущий уровень стека может оказаться разным в зависимости от того, откуда было передано управление на начало базового блока.

Одно из возможных решений — согласование уровня стека при помощи инструкций `push` и `pop`. Каждый переход на начало базового блока должен будет передавать управление сначала на последовательность согласующих команд `push` или `pop`, и только после согласования на начало базового блока.

Для более эффективного решения проблемы согласования уровня стека в систему команд Etherbox32vm была введена команда `adjust`. Команда `adjust` получает один аргумент на стеке: число элементов стека относительно начала текущего стекового фрейма, которое должно остаться на стеке после выполнения этой команды. Если в текущий момент на стеке находятся избыточные элементы, они удаляются. Если элементов недостаточно, добавляются нулевые элементы в нужном количестве.

Команда `adjust` добавляется компилятором языка Etherbox2 перед началом каждого базового блока.

2.9. Работа со внешними интерфейсами

В рассматриваемой сенсорной сети узлы имеют модульную конструкцию. Все датчики и исполнительные механизмы подключены

[3]Базовый блок — последовательность инструкций, содержащая единственную точку входа (начало блока), единственную точку выхода (конец блока) и не содержащая инструкций передачи управления за границы блока.

Таблица 2. Структура команды `i2c`

Поле	`opcode`	`len`	`count`	`data`
Длина *в байтах*	1	1	2	*len*

к модулю, где работает Etherbox32vm, посредством шины `I²C` [15]. Доступ к устройствам на шине осуществляется одноименной командой i2c. Структура команды i2c показана в таблице 2.

В первом байте поля data записывается адрес устройства на шине, с которым осуществляется транзакция. В зависимости от значения младшего бита адреса, осуществляется операция чтения или записи на двухпроводном интерфейсе.

По окончании операции чтения в поле ocount записывается количество успешно считанных байтов, а сами данные, полученные в результате чтения, записываются в поле data. Аналогично, в ходе операции записи поле data служит источником передаваемых данных.

Подобным же образом, с сохранением результаов операций чтения в теле проглета, устроены команды работы с интерфейсами UART и CAN Bus.

2.10. Групповой опрос сенсорных узлов

Одной из типовых операций, совершаемых в процессе эксплуатации сенсорной сети, является периодический опрос сенсорных узлов. Опрос со стороны управляющей станции не является лучшим способом получения информации от сенсорных узлов, чаще применяется асинхронная отправка данных сенсорными узлами. Тем не менее, централизованный периодический опрос может использоваться как для получения информации, так и для проверки реакции сенсорных узлов на команды управляющей станции (реализация сторожевого таймера — команда `wdt`[4]).

Для опроса узлов со стороны управляющей станции отправляется пакет с групповой (multicast) адресацией, содержащий команды считывания состояния узла и команду `wdt`. Но конфигурация узлов в сенсорной сети может различаться, соответственно различается и набор данных, составляющих статус каждого сенсорного узла, а значит должен различаться и набор действий, выполняемых сенсорным узлом

[4] от слов watchdog timer (англ.)

для составления статуса. То есть в общем случае составить проглет, подходящий для всех узлов при групповом опросе, невозможно.

Для решения этой проблемы в системе команд Etherbox32vm присутствует полиморфная команда `poll`, по которой каждый узел отрабатывает установленную во время конфигурации индивидуальную последовательность команд. Последовательность команд, выполняемая по команде `poll`, сохраняется командой `savepoll`.

2.11. Сохранение проглетов

В разделе упоминалось, что при включении сенсорный узел не имеет сценария работы и ждет конфигурации (сценария) от управляющей станции. Но бывают ситуации, когда узлу необходим начальный сценарий немедленно после включения. Например, нужно привести исполнительный механизм в исходное положение, даже если связность сенорной сети нарушена. Или для подключения к транспортной сети требуются настройки (например, Wi-Fi).

Для таких случаев предусмотрена команда `savepacket`, сохраняющая текущий проглет в энергонезависимой памяти устройства. Одновременно в устройстве может быть сохранено несколько проглетов. После перезагрузки устройства сохраненные проглеты автоматически загружаются и начинают исполнение с инструкции, следующей за инструкцией `savepacket`. Команда `savepacket` возвращает булевское значение, по которому проглет может отличить ситуацию «проглет успешно сохранен» от ситуации «сохраненный проглет загружен на исполнение».

2.12. Обработка ошибок

По умолчанию, ошибки времени исполнения (например, доступ к несуществующему адресу памяти, переполнение стека, вызов несуществующей публичной функции) приводят к аварийному прекращению исполнения проглета с последующей отправкой ответа на управляющую станцию (см. раздел 2.13).

Это поведение можно изменить при помощи команды `catch`. Команда `catch` включает в текущей функции перехват ошибок. Если в ходе выполнения функции (или вызванных из нее других функций) после команды `catch` возникает ошибка, исполнение проглета не прекращается, как обычно. Вместо этого происходит возврат из функции, выполнившей `catch`, причем в качестве возвращаемого функцией

значения используется значение, переданное в качестве аргумента команде `catch`. Это значение выбирают так, чтобы оно отличалось от нормальных значений, возвращаемых функцией: тогда по возвращенному функцией значению можно будет понять, завершилась она нормально или в результате перехвата ошибки.

Выбор возврата из функции в качестве реакции на возникновение ошибки обусловлен простотой реализации: восстановление от ошибки в любой точке функции, при любом состоянии стека сводится к удалению со стека всех переменных и временных значений, добавленных текущей функцией. Необходимый для этого уровень начала стекового фрейма текущей функции уже присутствует в стеке возвратов. А попытка поддержать конструкцию `try..catch` в духе языка Си++ потребовала бы специального сохранения состояния стека и адреса catch-блока.

2.13. Завершение проглета и отправка ответа

Завершение работы проглета происходит в следующих случаях:

(*1*) исполнение достигает конца программы;
(*2*) в результате исполнения команды end;
(*3*) в результате ошибки времени исполнения.

Завершившийся проглет удаляется из памяти сенсорного узла и целиком передается обратно на управляющую станцию в том состоянии, в котором он находился на момент окончания исполнения.

Есть также возможность отправить сообщение на управляющую станцию и не завершая работу проглета. Команда `send` передает текущий проглет на управляющую станцию, а затем продолжает его исполнение.

Часто передавать весь проглет оказывается избыточным. Например, проглет может производить в цикле опрос периферийных устройств и передавать полученные от них данные на управляющую станцию. Передавать весь проглет при этом необязательно: достаточно передать фрагмент проглета, содержащий команду `i2c`, в теле которой находится ответ опрашиваемого устройства.

Команда `send3` позволяет отправить на управляющую станцию произвольный фрагмент текущего проглета. Адрес начала и длина передаваемого фрагмента являются аргументами команды `send3`.

2.14. Ограничения

В разделе 2.3 сказано, что Etherbox32vm реализует набор инструкций, достаточный для реализации Си-подобного языка. Это действительно так, но язык будет иметь несколько существенных ограничений по сравнению с языком Си:

- общий объем задействованных аргументов функций и автоматических переменных в любой момент выполнения программы ограничен глубиной стека виртуальной машины, которая на малых микроконтроллерах совсем небольшая (64 слова). Поэтому рекурсивные приемы программирования практически неприменимы, и при создании автоматических переменных приходится соблюдать осторожность, чтобы не столкнуться с переполнением стека во время выполнения;
- поскольку стек находится вне адресного пространства проглета, невозможно получить указатель на автоматическую переменную. Это особенно заметно для автоматических массивов: доступ по индексу реализуем, доступ по указателю принципиально нет;
- беззнаковые (unsigned) типы не поддерживаются. Это не принципиальное ограничение, при необходимости их можно поддержать ценой добавления команд беззнакового сравнения;
- арифметика с плавающей точкой не поддерживается. На современных малых микроконтроллерах нет поддержки операций с плавающей точкой, потребовалось бы использовать их программную реализацию, что неоправданно утяжелило бы виртуальную машину.

2.15. Примеры программ

Ниже приведены три проглета, написанные на языке ассемблера Etherbox32vm. Стоит отметить, что в эксплуатации для кодирования проглетов обычно используется язык более высокого уровня, Си-подобный язык Etherbox2, компилируемый в байт-код Ethrebox32vm.

2.15.1. *Пример 1*

Простейший проглет, инкрементирующий значение, находящееся в сегменте данных. Сегмент данных здесь имеет длину один байт, содержащий значение 33. По завершении проглета, проглет будет возвращен на управляющую станцию со значением 34 в теле команды `data` взамен первоначального 33.

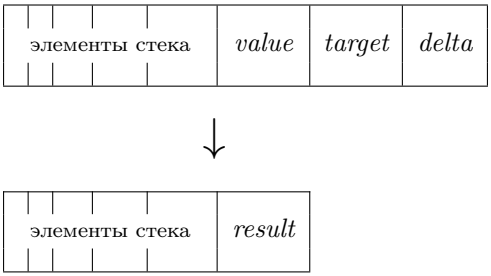

Рис. 5. Стек до и после команды `near`

```
data:
  data 33              | сегмент данных проглета
  load8 data+3+0       | загрузка значения 33 на стек
  push 1               |
  add                  |
  store8 data+3+0      | сохранение результата (34) обратно в data
```

Стоит заметить, что сегмент данных находится на смещении 3 относительно адреса самой команды `data` — см. описание структуры команды `data` в таблице 1.

2.15.2. *Пример 2*

Проглет, опрашивающий в цикле с периодом в 0.5 секунды периферийное устройство через интерфейс I^2C. В ответе устройства находятся, среди прочего, показания датчика температуры. В сегменте данных находятся переданные управляющей станцией номинальное значение температуры и допустимое отклонение измеренной температуры от номинального в единицах АЦП. Проглет завершается, если измеренная температура отличается от номинальной на величину, превышающую допустимое отклонение.

Команда near получает три аргумента на стеке: *value*, *target* и *delta*, и возвращает 1 в случае, если *value* отличается от *target* на величину меньшую, чем *delta* (рис. 5).

Команда `i2c` в этом примере осуществляет чтение 12 байт с устройства с адресом 37 на шине I^2C. Завершившая чтение команда `i2c`

`data:`	
`  data 132 15 232 15`	0 градусов (16 бит: 15*256+132=3972), 1 градус (16 bit: 15*256+232=4072) в единицах АЦП датчика температуры
`read:`	
`  i2c r37/12`	чтение показаний периферийного устройства
`  mspause 500`	задержка 0.5 секунды
`  load16 read+3+2`	полученная от датчика температура
`  load16 data+3+0`	целевое значение температуры
`  load16 data+3+2`	допустимая величина ошибки
`  near`	
`  push read`	
`  breq`	

выглядит в байткоде так:

$$\underbrace{15}_{opcode}\ \underbrace{13}_{len}\ \underbrace{12}_{count}\ \underbrace{37\ \ 0\ \ 107\ \ 28\ \ 50\ \ 2\ \ 0\ \ 0\ \ 0\ \ 61\ \ 134\ \ 124\ \ 250}_{addr,\,data}$$

Показания датчика температуры указаны в единицах АЦП температурного датчика и представлены 16-битным значением на смещении 1 от начала ответа в порядке little endian. Здесь это байты со значениями 107 и 28.

2.15.3. *Пример 3*

Проглет, опрашивающий в бесконечном цикле с периодом в 10 секунд датчик температуры. Здесь проверка полученного значения отсутствует. Вместо этого, ответ периферийного устройства передается целиком на управляющую станцию командой `send3`. В отличие от предыдущего примера, в ответе оказывается не весь проглет, а только команда `i2c`, в теле которой содержится ответ периферийного устройства. Такой ответ тоже является проглетом, состоящим из единственной команды, и декодируется на управляющей станции по общим правилам.

`read:`	
`i2c r37/12`	чтение показаний периферийного устройства
`send3 12+1+3 read 0`	отправка на управляющую станцию участка проглета длиной 16, начинающегося с метки read
`dspause 100`	задержка 10 секунд
`push read`	
`br`	

Заключение

Представленная в статье архитектура виртуальной машины является результатом эволюции Etherbox32vm под влиянием опыта практического использования в сенсорных сетях. Использование виртуальной машины позволяет достичь поставленной цели — безопасного удаленного изменения поведения узлов сенсорной сети. В форме команд виртуальной машины действительно удается с достаточной эффективностью выразить встречающиеся на практике сценарии функционирования сенсорных узлов.

При этом остается место для дальнейшего развития. До сих пор использовался только обмен проглетами между управляющей станцией и узлами; при этом управляющая станция компилирует или ассемблирует проглеты перед отправкой и дизассемблирует ответные проглеты, чтобы извлечь добавленную узлами информацию. Одним из направлений дальнейшей работы является организация прямого взаимодействия между узлами (например, передача информации от узла, обслуживающего датчик, узлу, обслуживающему зависящий от датчика исполнительный механизм). Есть надежда организовать такое взаимодействие в терминах проглетов без введения дополнительного протокола.

Список литературы

[1] URL: https://raspberrypi.org/ ↑ 120

[2] URL: https://www.olimex.com/Products/OLinuXino/open-source-hardware ↑ 120

[3] A. Dunkels, et al. “Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks”, *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys’06 (October 31–November 03, 2006, Boulder, Colorado, USA), ACM, New York, USA, 2006, pp. 15–28. ↑ 122,123

[4] Contiki: The Open Source OS for the Internet of Things, URL: http://www.contiki-os.org/ ↑ 122

[5] J. Koshy, R. Pandey. “VMSTAR: Synthesizing Scalable Runtime Environments for Sensor Networks”, *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys’05 (November 02–04, 2005, San Diego, California, USA), ACM, New York, USA, 2005, pp. 243–254. ↑ 122

[6] T. Lindholm, F. Yellin. *The Java Virtual Machine Specification*, Addison-Wesley, 1999. ↑ 122

[7] P. Levis, D. Culler. “Maté: A Tiny Virtual Machine for Sensor Networks”, *Proceedings of Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS’02 (October 05–09, 2002, San Jose, CA, USA), ACM, New York, USA, 2002, pp. 85–95. ↑ 122

[8] J. Postel. *User Datagram Protocol*, RFC768, RFC Editor, August 28, 1980, URL: http://www.rfc-editor.org/rfc/rfc768.txt ↑ 123

[9] A. A. Milne. *Winnie-the-Pooh*, Methuen & Co. Ltd., London, 1926. ↑ 124

[10] Y. Shi, et al. “Virtual Machine Showdown: Stack Versus Registers”, *ACM Trans. Archit. Code Optim.*, **4**:4 (January 2008), Article No. 2. ↑ 125

[11] P. Koopman, *Stack Computers*, Ellis Horwood Series in Computers and Their Applications, Ellis Horwood/Halstead Press, 1989, 236 p. ↑ 125

[12] P. J. Koopman, Jr. “A Preliminary Exploration of Optimized Stack Code Generation”, *Journal of Forth Application and Research*, **6**:3 (1994), pp. 241–251. ↑ 125

[13] M. Shannon, C. Bailey. “Global Stack Allocation (Register Allocation for Stack Machines)”, *Procedings of 22nd EuroForth Conference*, EuroForth 2006 (September 15–17, 2006, Cambridge, England), 2006, 8 p., URL: http://www.complang.tuwien.ac.at/anton/euroforth2006/papers/shannon.pdf ↑ 125

[14] А. Шевчук. «Компилятор языка Etherbox2», *Наукоемкие информационные технологии* (Переславль-Залесский, 2010), 9 с., URL: http://site.u.pereslavl.ru/Studentu/Konkursy/konferenciya/2010/091-10.pdf ↑ 127

[15] *UM10204. I^2C-bus specification and user manual*, NXP Semicondictor, 2014, 64 p., URL: http://www.nxp.com/documents/user_manual/UM10204.pdf ↑ 133

Рекомендовал к публикации *к.ф.-м.н. А.В. Климов*

Пример ссылки на эту публикацию:

Ю. В. Шевчук, А. Ю. Шевчук. «Виртуальная машина «Etherbox32vm»», *Программные системы: теория и приложения*, 2016, **7**:4(31), с. 119–143.
URL: http://psta.psiras.ru/read/psta2016_4_119-143.pdf

Об авторах:

Юрий Владимирович Шевчук
Зав. лабораторией телекоммуникаций, к.т.н. Область интересов: системное программирование, цифровая электроника, сети компьютеров, сенсорные сети.
e-mail: sizif@botik.ru

Андрей Юрьевич Шевчук
Инженер-исследователь. Область интересов: программирование встроенных систем, Web-программирование
e-mail: shadoy@sand.botik.ru

Yury Shevchuk, Andrey Shevchuk. *Etherbox32vm virtual machine.*

Abstract. The article presents the Etherbox32vm virtual machine architecture. Etherbox32vm is a virtual machine implemented on network nodes in heterogeneous sensor networks to make it possible to change node behaviour remotely. Sensor aquisition and actuator control scenarios, as well as preliminary sensor data processing, can be specified in the form of small programs for the Etherbox32vm virtual machine. Etherbox32vm is light-weight enough to be implementable on microcontrollers with scarce RAM. (*In Russian*).

Key words and phrases: sensor networks, virtual machine, stack machine.

References

[1] URL: https://raspberrypi.org/

[2] URL: https://www.olimex.com/Products/OLinuXino/open-source-hardware

[3] A. Dunkels, et al. "Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks", *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys'06 (October 31–November 03, 2006, Boulder, Colorado, USA), ACM, New York, USA, 2006, pp. 15–28.

[4] Contiki: The Open Source OS for the Internet of Things, URL: http://www.contiki-os.org/

[5] J. Koshy, R. Pandey. "VMSTAR: Synthesizing Scalable Runtime Environments for Sensor Networks", *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys'05 (November 02–04, 2005, San Diego, California, USA), ACM, New York, USA, 2005, pp. 243–254.

[6] T. Lindholm, F. Yellin. *The Java Virtual Machine Specification*, Addison-Wesley, 1999.

[7] P. Levis, D. Culler. "Maté: A Tiny Virtual Machine for Sensor Networks", *Proceedings of Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'02 (October 05–09, 2002, San Jose, CA, USA), ACM, New York, USA, 2002, pp. 85–95.

[8] J. Postel. *User Datagram Protocol*, RFC768, RFC Editor, August 28, 1980, URL: http://www.rfc-editor.org/rfc/rfc768.txt

[9] A. A. Milne. *Winnie-the-Pooh*, Methuen & Co. Ltd., London, 1926.

[10] Y. Shi, et al. "Virtual Machine Showdown: Stack Versus Registers", *ACM Trans. Archit. Code Optim.*, **4**:4 (January 2008), Article No. 2.

[11] P. Koopman, *Stack Computers*, Ellis Horwood Series in Computers and Their Applications, Ellis Horwood/Halstead Press, 1989, 236 p.

[12] P. J. Koopman, Jr. "A Preliminary Exploration of Optimized Stack Code Generation", *Journal of Forth Application and Research*, **6**:3 (1994), pp. 241–251.

[13] M. Shannon, C. Bailey. "Global Stack Allocation (Register Allocation for Stack Machines)", *Procedings of 22nd EuroForth Conference*, EuroForth 2006 (September 15–17, 2006, Cambridge, England), 2006, 8 p., URL: http://www.complang.tuwien.ac.at/anton/euroforth2006/papers/shannon.pdf

[14] A. Shevchuk. “Etherbox2 language compiler”, *Proceedings of Junior research and development conference of Ailamazyan Pereslavl University* (Pereslavl-Zalessky, 2010), pp. 91–99 (in Russian), URL: http://site.u.pereslavl.ru/Studentu/Konkursy/konferenciya/2010/091-10.pdf

[15] *UM10204. I^2C-bus specification and user manual*, NXP Semicondictor, 2014, 64 p., URL: http://www.nxp.com/documents/user_manual/UM10204.pdf

Sample citation of this publication:

Yury Shevchuk, Andrey Shevchuk. “Etherbox32vm virtual machine”, *Program systems: Theory and applications*, 2016, **7**:4(31), pp. 119–143. (*In Russian*). URL: http://psta.psiras.ru/read/psta2016_4_119-143.pdf