Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Доледенок Максим Вадимович

Разработка средств активного наблюдения за состоянием бортовой операционной системы реального времени

Выпускная квалификационная работа

Научный руководитель:
к.ф.-м.н.
Камкин Александр Сергеевич

Научный консультант:
Чепцов Виталий Юрьевич

Москва, 2023

Аннотация

Разработка средств активного наблюдения за состоянием
бортовой операционной системы реального времени

Доледенок Максим Вадимович

При эксплуатации бортовой операционной системы реального времени, предназначенной для применения на космических аппаратах, в программно-аппаратном комплексе должно возникать как можно меньше непредвиденных ситуаций. Одна из возможностей минимизировать количество таких ситуаций – это предварительная отработка полетного задания на программно-аппаратном комплексе в более контролируемых условиях на земле. Для большего контроля над системой во время отработки полётного задания требуется регулярно получать телеметрическую информацию изнутри системы во время её работы. Таким образом, запуская тестовые и реальные сценарии на ОСРВ, в инструменте можно видеть изменения состояния объектов в реальном времени и отслеживать аномалии. Также для моделирования исключительных ситуаций необходима возможность в реальном времени изменять состояние программного обеспечения.

В данной работе рассматривается метод разработки инструмента для доступа к памяти системы на примере бортовой операционной системы реального времени с поддержкой стандарта ARINC 653.

Abstract

Doledenok Maxim Vadimovich

When operating an on-board real-time operating system intended for use on spacecraft, there should be a problem in the hardware and software complex as few unforeseen situations as possible. One of the possibilities is to minimize the number of such situations are preliminary testing of a flight task on a software and hardware complex in more controlled conditions on the ground. For greater control over the system during the flight task, it is required to receive telemetry information on a regular basis from inside the system during its operation. Thus, running test and real-world scenarios on the RTOS, in the tool you can see changes in the state of objects in real time and track anomalies. Also, for modeling exceptional situations, it is necessary to be able to change the state of the software in real time.

This paper discusses the method of developing a tool for accessing system memory using the example of an on-board real-time operating system with support for the ARINC 653 standard.

Содержание

1 Введение

1.1 Операционные системы реального времени

Операционные системы реального времени (ОСРВ) – это операционные системы, одним из важнейших требований к которым является выполнение поставленных задач за заранее определенное время. Это отличает их от известных пользовательских операционных систем, где фактор времени не так важен. ОСРВ используются там, где небольшая временная задержка может привести к существенным проблемам. Например, в авиакосмической отрасли, на некоторых производствах, в системах аварийной защиты.

1.2 Бортовая операционная система реального времени с поддержкой ARINC 653

В данной работе рассматривается бортовая операционная система реального времени (БОСРВ), разрабатываемая в институте системного программирования им. В.П. Иванникова Российской академии наук. БОСРВ предназначена для использования в космических аппаратах. Данная БОСРВ разрабатывается в соответствии со стандартом ARINC 653 [2], который регламентирует временное и пространственное разделение ресурсов авиационной ЭВМ и определяет программный интерфейс, которым должно пользоваться прикладное ПО для доступа к ресурсам ЭВМ. Единицей планирования ресурсов является раздел, аналог пользовательской программы. Каждый раздел получает как процессорное время, так и некоторую часть оперативной памяти.

1.3 Инструмент для активного наблюдения за памятью БОСРВ

На программно-аппаратных комплексах бортовой ОСРВ количество непредвиденных ситуаций должно быть сведено к минимуму ввиду особых требований по надёжности ПО. Одна из возможностей минимизировать количество непредвиденых ситуаций – это предварительная отработка полетного задания на программно-аппаратном комплексе в более контролируемых условиях на земле. Проведение отладки в реальном времени посредством наблюдения за целевой системой изнутри позволяет обнаружить

аномалии, которые в иной ситуации могли быть не зафиксированы только посредством анализа внешних реакций. Например, выход некоторых переменных за границу допустимых значений, не приводящий к ошибке в конкретной ситуации, но потенциально фатальный при длительной работе. Также для сбора и последующего анализа работы системы необходимо собирать телеметрическую информацию изнутри, при этом внося минимальное возмущение в работу бортовой ОСРВ. Поэтому необходим инструмент, который будет в реальном времени получать доступ как к памяти разделов, так и памяти ядра ОСРВ.

Под активным наблюдением за памятью ОСРВ подразумевается, что инструмент должен не только получать данные из памяти разделов и ядра ОСРВ, но и иметь возможность в реальном времени изменять состояние программного обеспечения. Это необходимо для моделирования исключительных ситуаций, то есть для намеренного внесения неисправностей в работу системы с целью проследить за её поведением. Например, для моделирования повреждения оперативной памяти аппаратного комплекса от радиации можно намеренно изменять значения ячеек памяти. То есть инструмент должен уметь считывать и записывать данные по каким-либо адресам как в памяти разделов, так и в памяти ядра ОС. Также, чтобы отладка с помощью инструмента была удобной для инженера, инструмент должен иметь возможность получать значения заданных переменных по их именам.

2 Цель работы, постановка задач

Целью данной работы является разработка инструмента для наблюдения и внесения изменений в память системного и прикладного ПО бортовой ОСРВ на базе стандарта ARINC 653 в реальном времени.

Для достижения цели выделен следующий набор задач:

- Исследовать принципы организации памяти в ARINC 653 совместимых ОСРВ.
- Построить архитектуру инструмента активного наблюдения за памятью ARINC 653 совместимой ОСРВ.
- Разработать прототип инструмента в рамках существующей ОСРВ и выполнить его тестирование.

В инструменте требуется обеспечить возможность чтения и записи как памяти ядра ОСРВ, так и разделов. Для удобства пользования инструмент должен выдавать данные наблюдения не только через прямую адресацию переменных, но и по их именам. Сложные типы данных, такие как структуры и массивы, тоже должны поддерживаться инструментом. Также для максимального приближения к реальным условиям требуется минимальное вмешательство в работу ОСРВ.

3 Обзор существующих решений

Было найдено несколько аналогов нужного инструмента, но ни один из них не удовлетворяет всем требованиям по разным причинам. На таблице 1 показано сравнение существующих решений, а ниже приведено их краткое описание.

Инструмент	В реальном времени	Чтение значений переменных по имени	Может работать на бортовой ОСРВ института системного программирования
JTAG	—	+	+
VxWorks	±	+	—
Инструмент для ОСРВ RTEMS	+	+	—
Инструмент COM	+	+	—

Таблица 1: Сравнение существующих решений

3.1 JTAG

JTAG [3] является промышленным стандартом для тестирования печатных плат, а также для отладки программного обеспечения. Но на платах, которые эксплуатируются в реальных условиях на спутниках, часто нет возможности использовать JTAG. К тому же, для отладки с JTAG надо прерывать работу программы, а хотелось бы иметь возможность отлаживать систему в реальном времени.

3.2 VxWorks

Также есть несколько проприетарных инструментов с аналогичными функциями. Например, у ОСРВ VxWorks [4] есть широкие возможности для отладки в реальном времени. В частности, он может выводить значения переменных программы, как глобальных, так и локальных, показывать данные по любому адресу программы, изменять

значения памяти. Также у него есть все возможности отладчика GDB, такие как установка точки останова, дизассемблирование кода, вывод значений регистров, обратная трассировка и другие. Вывод отладочной информации реализован через графический интерфейс. Но отладка с VxWorks требует много ресурсов и времени, так как выводит в реальном времени много отладочной информации. Также для некоторых функций необходимо прерывать работу программы. Так что для тестирования в максимально приближенных к реальным условиях этот инструмент не очень подходит.

3.3 Инструмент РКК «Энергия»

У РКК «Энергия» есть свой подобный инструмент, написанный для операционной системы RTEMS [5]. Он в реальном времени может получать доступ ко всем глобальным переменным программы, выводить их в читаемом виде и логировать. Но он работает только на архитектуре MIPS, и написан целенаправленно для операционной системы RTEMS.

3.4 Диссертация «Организация контролируемого выполнения для разнородных распределенных программно-аппаратных комплексов»

В диссертации Костюхина Константина Александровича «Организация контролируемого выполнения для разнородных распределенных программно-аппаратных комплексов» [6] подробно описан аналогичный инструмент COM – Система отладки/мониторинга. Он имеет широкие возможности для отладки, может наблюдать за памятью системы в реальном времени, показывать дизассемблированнй код. Он работает на ОС Linux, и на ОС реального времени ос2000. Его интерфейс можно видеть на рис. 1.

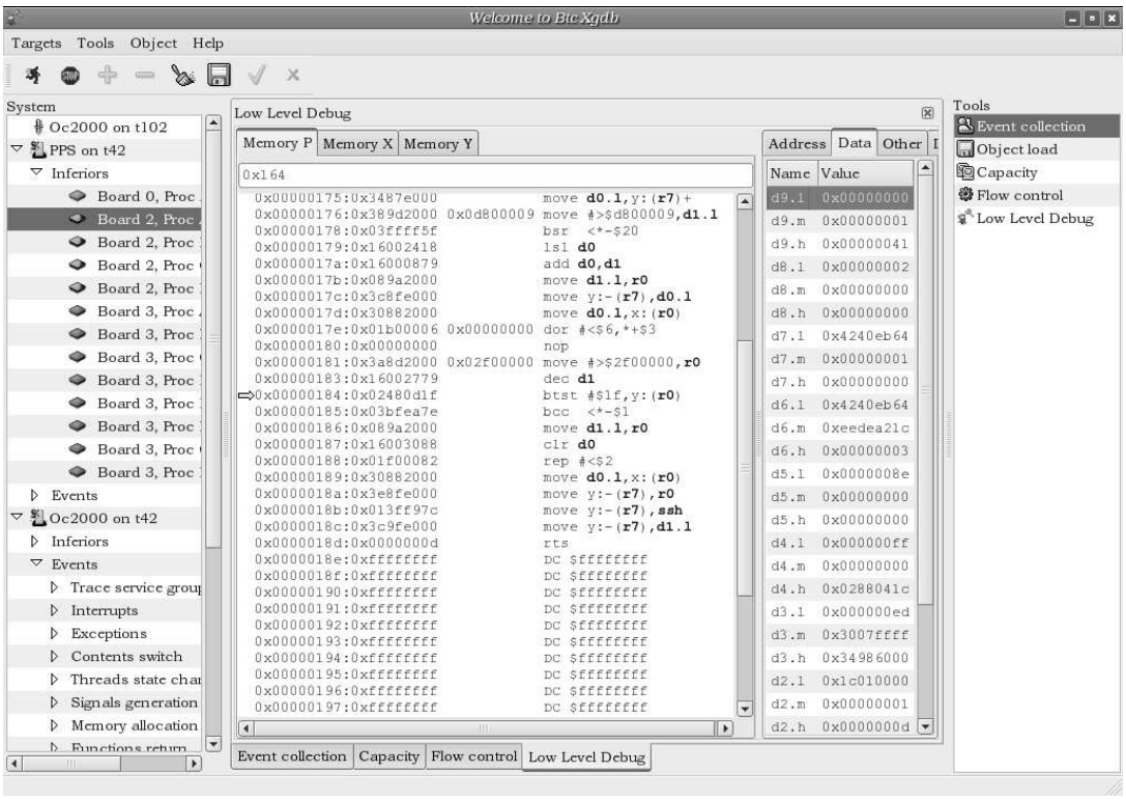

Рис. 1: Графический интерфейс инструмента COM

4 Исследование и простроение решения задачи

4.1 Устройство памяти в БОСРВ

В БОСРВ конфигурация памяти является статической, то есть раскладка памяти происходит на этапе загрузки операционной системы, исходя из заранее описанной пользователем конфигурации. При динамической конфигурации памяти пользовательские приложения могут заимствовать области оперативной памяти для временного использования. Но для ОСРВ это не всегда подходит, потому что сложно обеспечить детерминированность при работе с памятью. А для БОСРВ детерминированность является очень важным фактором, повышающим безопасность и отказоустойчивость операционной системы. Поэтому в БОСРВ применяется строго статическая конфигурация памяти. Причем, согласно стандарту ARINC 653 [2], каждый раздел имеет своё адресное пространство, такое, что никакой другой раздел не имеет доступа к памяти этого адресного пространства. А ядро БОСРВ имеет доступ к памяти всех разделов с такими же правами, что и сами разделы.

Основные принципы устройства памяти в БОСРВ:

- платформа имеет одно или несколько ядер процессора;
- набор регионов физической памяти общий для всех ядер;
- на одной платформе может быть запущено несколько модулей, здесь модуль – сконфигурированная операционная система, предназначенная для выполнения на одном или нескольких ядрах процессора на платформе;
- память разделяется на виртуальную и физическую;
- виртуальная память своя для каждого модуля, физическая – общая для всех модулей;
- виртуальная память разделяется на привилегированную (доступную только ядру) и непривилегированную (доступную и ядру, и разделам);
- физическая память разделяется на оперативную память и память физических устройств.

4.2 Схема работы инструмента

Исходя из организации памяти бортовой ОСРВ и требований к инструменту наблюдения, была представлена схема работы инструмента, которую можно посмотреть на рис. 2. Агент внутри системы, работающий в системном разделе, в нужный ему момент времени вызывает функцию из библиотеки, которая использует системный вызов, управление передается ядру, в котором происходит сначала обработка запросов от внешнего инструмента (если запросы были). Затем происходит чтение данных для наблюдения по адресам в разных адресных пространствах (поскольку код работает на уровне ядра, то есть доступ к памяти). И потом эти данные передаются во вне через UART. Внешний инструмент считывает эти данные и их логирует. Затем он обрабатывает запросы от пользователя, если такие были, формирует запросы для библиотеки, используя ELF-файлы, и отправляет их.

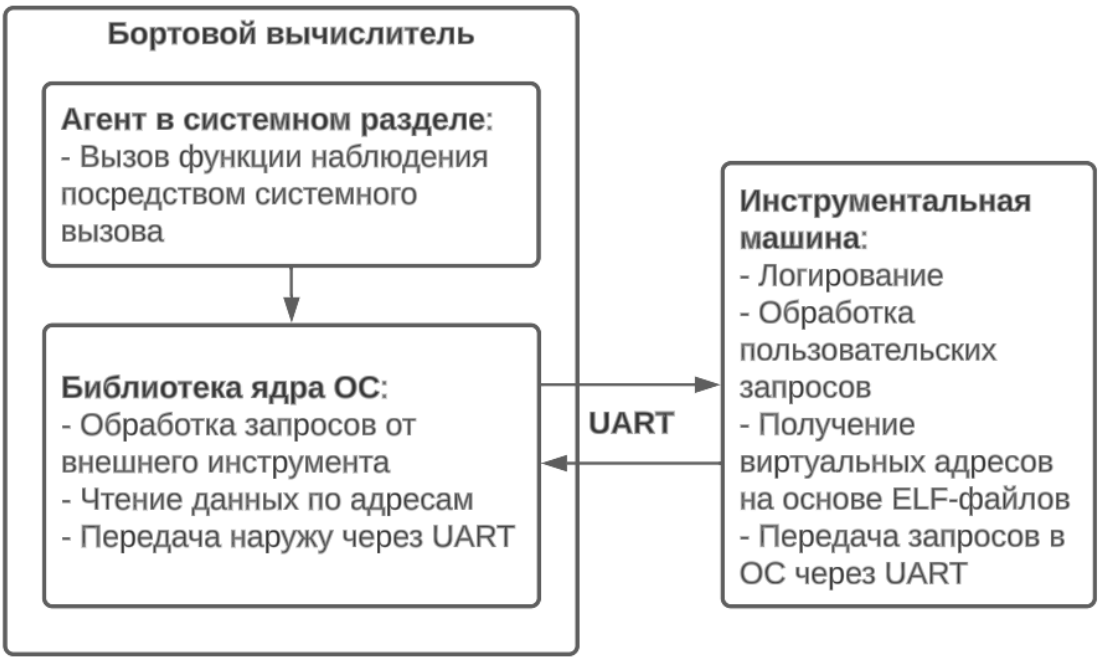

Рис. 2: Схема работы инструмента для наблюдения

4.3 Функциональность инструмента

Реализованному инструменту можно задавать запросы на чтение или изменение:

- переменной;
- данных конкретного размера по конкретному адресу;
- элемента структуры;
- элемента массива.

Список задаваемых для остлеживания переменных формируется двумя способами, статически через файл в корневой директории проекта, или динамически во время работы программы через консоль. Инструмент же выводит результаты отслеживания с временными метками в файл логирования.

4.4 Задание переменных для отслеживания

4.4.1 Задание переменных статически в файле YAML [1]

Перед запуском проекта в корневой папке проекта можно создать файл *monitor.yaml*. Файл представляет собой словарь вида ключ-значение. Ключевое слово `kernel` (необязательное) задаёт список переменных из ядра для наблюдения/изменения. Далее ключевыми словами являются имена разделов (необязательные), которые задают список переменных из соответствующих разделов для наблюдения/изменения. Каждая переменная задаётся следующими полями:

- `name` – имя переменной. Может отсутствовать, только если задан параметр `address`.
- `address` – адрес, по которому мы хотим считывать/записывать. Указывается, если не указан параметр `name`.
- `value` – значение, которое нужно записать в переменную `name`, или по адресу `address`. При указании этого поля переменная не отслеживается, просто изменяется значение. Для её отслеживания надо записать отдельно без поля `value`.
- `size` – размер данных, который надо считать/записать. Если не указывать это поле, то размер берётся из таблицы символов ядра или разделов.
- `type` — тип переменной. Может принимать значения:

- `pointer` – для получения/изменения значения по указателю, который лежит в переменной.
- `array` – для получения/изменения значения элемента массива.
- `struct` – для получения/изменения значения поля структуры.

- `elem` – указывается только если поле `type` имеет значения `array` или `struct`. Для массива указывается индекс элемента, который надо считать или изменить. Для структуры надо указать имя поля, которое надо считать или изменить.
- `format` – формат вывода переменной. Задаётся размер в байтах. Значение переменной печатается частями по этому количеству байт через пробел.

Пример файла *monitor.yaml*:

```
kernel:
   - name: sPartitionManagerPartitionIndicesNumber
P1:
   - name: part_var
   - name: part_var
     value: 0xff
   - name: part_array
     type: array
     elem: 3
   - name: part_ptr
     type: pointer
     size: 4
   - name: part_struct
     type: struct
     elem: second_elem
```

4.4.2 Задание переменных динамически через консоль

Также можно вносить изменения в список отслеживания, или изменять значения переменных в реальном времени. Для этого надо в терминале, из которого был запущен инструмент, записать команду в формате "`name=... type=... ; name=... value=... ; name=...`". Дополнительно можно указать поле delete=1 для удаления переменной из отслеживания.

4.5 Протокол взаимодействия инструментальной машины и бортового вычислителя

Инструментальная машина и бортовой вычислитель обмениваются сообщениями через UART. UART (Universal Asynchronous Receiver-Transmitter) – узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. UART представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения. Для максимизации скорости передачи данных, лучше всего пересылать данные не в строковом формате, а в бинарном, поскольку данные в бинарном формате занимают намного меньше памяти, чем данные в строковом формате.

4.5.1 Протокол посылки команд бортовому вычислителю

Инструмент посылает команды операционной системе, если пользователь ввел запрос на отслеживание/изменение новой переменной в бинарном формате. Также при запуске проекта формируется блок памяти на основе файла *monitor.yaml* с таким же форматом данных. Команды имеют вид: "`<управляющий байт> <адрес> <размер> <значение>`".

- `управляющий байт` содержит флаги:
    - Биты [4:0] передают номер адресного пространства, в котором надо отследить/изменить переменную.
    - Биты [6:5] отвечают за тип команды:
        - Значение 0 означает отслеживание/изменение переменной.
        - Значение 1 означает отслеживание/изменение переменной по указателю, который надо разыменовать.
        - Значение 2 означает удаление переменной из списка отслеживания.
    - Биты [4:0] передают номер адресного пространства, в котором надо отследить/изменить переменную.

- Биты [6:5] отвечают за тип команды:
  - Значение 0 означает отслеживание/изменение переменной.
  - Значение 1 означает отслеживание/изменение переменной по указателю, который надо разыменовать.
  - Значение 2 означает удаление переменной из списка отслеживания.
  - Значение 3 означает конец передачи команд отслеживания для текущего пакета.
- Бит [7:7] устанавливается равным 1, если надо изменить переменную. Иначе 0.

- `адрес` – 4 или 8 байт в зависимости от архитектуры.
- `размер` отслеживаемой/изменяемой переменной. Размер занимает 1 байт.
- `значение` длиной в `размер` байт. Указывается только если установлен первый бит `управляющего байта`.

4.5.2 Протокол посылки данных целевой системе

Библиотека ядра ОСРВ посылает данные инструментальной машине вне ОС также через UART.

Для каждого адресного пространства посылается один байт с номером этого пространства, затем посылается один байт с количеством отслеживаемых переменных для этого пространства, далее посылаются 4 или 8 байт, в зависимости от архитектуры, представляющие собой адрес, по которому отслеживается переменная. Далее идет значение переменной, размер которой может варьироваться от 1 до 255 байт. Далее опять посылается адрес переменной, и её значение.

4.6 Получение адреса переменной по её названию

Для получения адреса переменной по её названию используются файлы формата ELF [7] с отладочной информацией DWARF [8]. ELF (Executable and Linking Format) – это формат исполняемых двоичных файлов, используемый во многих современных

UNIX-подобных операционных системах, таких как Linux, FreeBSD и прочих. С помощью этого файла можно получить адреса глобальных переменных программы, их тип, размер и прочее.

Но получить подробную информацию о переменных сложных типов не получится, в файле формата ELF эта информация не хранится. Данные о переменных сложных типов можно получить из отладочной информации DWARF. DWARF - это широко используемый стандартизированный формат отладочных данных. Изначально DWARF был разработан вместе с форматом ELF, хотя он не зависит от форматов объектных файлов. Для лучшей отладки программу можно скомпилировать с отладочной информацией DWARF, и получить подробную информацию о всех переменных, функциях и типах данных, используемых в программе. В частности, в отладочной информации DWARF хранится информация о всех глобальных структурах, массивах и других данных сложных типов, таким образом, можно получать доступ к полям структур, и элементов массивов.

Часть инструмента, расположенная на инструментальной машине, была написана на языке Python, поэтому целесообразно использовать уже имеющиеся библиотеки Python, написанные для чтения ELF файлов. Была выбрана библиотека pyelftools [9].

4.6.1 Переменные простых типов

Для получения адресов и размеров переменных простых типов достаточно двоичного файла программы в формате ELF [7]. Каждый ELF файл состоит из заголовка файла, таблицы заголовков программ, таблицы заголовков секций и содержимого секций и сегментов. В заголовке ELF файла хранятся основные характеристики файла, такие как тип, версия формата, архитектура процессора, виртуальный адрес точки входа, размеры и смещение остальных частей файла. Каждый заголовок из таблицы заголовков программ описывает отдельный сегмент программы и его атрибуты, либо другую информацию, необходимую операционной системе для подготовки программы к исполнению. В таблице заголовков секций содержатся атрибуты секций файла. Секции файла нужны только для компоновщика программы.

Информация о переменных раздела хранится в секции .symtab. Библиотека pyelftools [9] предоставляет удобный интерфейс для получения информации о символах программы. В частности, для каждого символа программы доступны следующие

атрибуты:

- `st_name` – индекс в массиве строк, указывающий на название символа.
- `st_value` – виртуальный адрес переменной.
- `st_size` – размер переменной.
- `st_info` – информация о символе, является ли он функцией, переменной или секцией, а также другая вспомогательная информация.
- `st_shidx` – индекс секции в таблице заголовков секций, в которой содержится данный символ.
- `st_other` – информация о видимости символа и некоторая другая вспомогательная информация.

Таким образом, по имени переменной можно узнать всю необходимую информацию для работы инструмента с переменными простых типов – их виртуальный адрес и размер символа. Для переменных сложных типов в ELF файле хранится только информация о самой переменной, её адрес и размер. Для структуры, например, нет возможности узнать адрес какого-нибудь её поля. А для массива нельзя узнать тип хранящихся в нём данных.

4.6.2 Переменные сложных типов

Для получения информации о переменных сложных типов, таких как структуры и массивы, лучше пользоваться отладочной информацией DWARF [8]. По умолчанию при сборке программы отладочная информация не собирается, но для отладки её можно собрать с отладочной информацией. DWARF использует структуру данных, называемую DIE (Debugging Information Entry), для представления каждой переменной, типа, процедуры и т.д. Эта структура имеет тег, обозначающий тип структуры, и атрибуты (пары ключ-значение). DIE может иметь вложенные (дочерние) структуры, образующие древовидную структуру. Атрибут DIE может ссылаться на другую структуру в любом месте дерева — например, структура, представляющая переменную, будет иметь тег DW_AT_type, указывающий на структуру, описывающую тип переменной.

Например, для объявления массива *const char **argv* структуры DIE будут такие:

```
<1>: DW_TAG_variable
     DW_AT_name = argv
     DW_AT_type = <2>
<2>: DW_TAG_pointer_type
     DW_AT_byte_size = 4
     DW_AT_type = <3>
<3>: DW_TAG_pointer_type
     DW_AT_byte_size = 4
     DW_AT_type = <4>
<4>: DW_TAG_const_type
     DW_AT_type = <5>
<5>: DW_TAG_base_type
     DW_AT_name = char
     DW_AT_byte_size = 1
     DW_AT_encoding = unsigned
```

У библиотеки pyelftools [9] есть интерфейс для чтения отладочной информации DWARF, но обходить дерево структур DIE для поиска конкретной переменной или типа данных нужно самому. В ходе разработки инструмента этот функционал был реализован.

4.7 Вывод наблюдаемых данных в файл

Программа на инструментальной машине выводит данные, полученные от агента в системном разделе, в файл. Пример вывода данных в файл:

```
0.000| kernel:
0.001|    sPartitionManagerPartitionIndicesNumber: 1
0.001| P1:
0.001|    part_var32: 0
0.001|    part_var16: ffff
0.002|    part_var64: aaaabbbbccccddd0
0.002|    part_array: 200 0 0 0 0 f 0 0 0 0
0.003|    part_struct: 1 0 3 e8 0 1 86 a0

0.101| kernel:
0.101|    sPartitionManagerPartitionIndicesNumber: 1
0.101| P1:
0.101|    part_var32: 100
0.101|    part_var16: eeee
0.102|    part_var64: aaaabbbbccccddd0
0.102|    part_array: 200 0 0 5 0 f 0 0 0 0
0.102|    part_struct: 1 0 3 e8 6 1 86 a0
```

Здесь слева показано время от начала наблюдения в секундах, а справа сама отладочная информация.

4.8 Анализ производительности

Инструмент был проверен на нескольких архитектурах, в том числе на радиационно стойком процессоре ЭЛВИС ВМ15АФ. Были произведены замеры времени работы инструмента, которые видны на рис. 3:

Количество разделов	Количество переменных размером 32 бита	Среднее время вывода значений всех переменных, мс
1	10	32
1	20	45
2	10	37
2	20	49
3	10	47
3	20	57

Рис. 3: Замеры времени одного цикла работы инструмента для ЭЛВИС ВМ15АФ

5 Результаты работы

В ходе работы был разработан инструмент, который в реальном времени может считывать и изменять память как прикладного, так и системного ПО бортовой ОСРВ. Реализованный инструмент является архитектурно-независимым, то есть может работать на любой архитектуре, которую поддерживает бортовая ОСРВ, разработанная в институте системного программирования им. В.П. Иванникова Российской академии наук. В частности, инструмент был проверен на радиационно стойком процессоре ЭЛВИС ВМ15АФ, который активно применяется в реальных условиях. Также инструмент обладает следующими характеристиками:

- Доступны запросы на чтение/запись переменных и данных по конкретному адресу в разделах и ядре.
- Доступны запросы на чтение/запись элемента структуры или массива.
- Возможность логирования наблюдаемых данных.
- Производительность инструмента достаточна для систем данного класса.

6 Заключение

В ходе выпускной квалификационной работы был разработан инструмент для наблюдения и внесения изменений в память операционной системы реального времени на базе стандарта ARINC 653, для этого было проведено исследование принципов организации памяти ОСРВ, предложена и реализована архитектура инструмента для наблюдения и внесения изменений в память прикладного и системного ПО операционной системы. В рамках реализованной архитектуры инструмент состоит из агента внутри операционной системы и программы на инструментальной машине. Агент считывает необходимые данные из памяти и отсылает их на инструментальную машину через UART. А программа на инструментальной машине обрабатывает запросы от пользователя, посылает команды агенту и выдаёт результаты наблюдения пользователю. Инструмент имеет возможность в реальном времени для системного и прикладного ПО читать и изменять значения любых переменных по их именам, причем как простых, так и сложных типов.

Разработанный инструмент делает более удобным отладку программы в реальном времени, и может быть полезным для разработчика программного обеспечения для бортовой ОСРВ. Также данный инструмент может являться основой для отладчика с любой графической оболочкой, чтобы ещё больше повысить удобство отладки ПО для бортовой ОСРВ.

Список литературы

[1] YAML Ain't Markup Language (YAML™) version 1.2.
https://yaml.org/spec/1.2.2/.

[2] ARINC Avionics Standards.
https://www.aviation-ia.com/product-categories/600-series/.

[3] JTAG Programmer Tutorial.
https://www.asc.ohio-state.edu/physics/cms/cfeb/datasheets/jtag.pdf.

[4] VxWorks Real Time Operating System.
https://resources.windriver.com/vxworks/vxworks-653-product-overview.

[5] RTEMS Real Time Operating System https://www.rtems.org/.

[6] Костюхин К. А. Организация контролируемого выполнения для разнородных распределенных программно-аппаратных комплексов: дис. канд. физ.-мат. наук / К. А. Костюхин – Москва, 2006. – 159 с.
https://www.dissercat.com/content/organizatsiya-kontroliruemogo-vypolneniya-dlya-raznorodnykh-raspredelennykh-programmno-appar.

[7] Executable and Linkable Format (ELF).
http://www.skyfree.org/linux/references/ELF$_F$*ormat.pdf*.

[8] DWARF Debugging Information Format Version 4.
https://dwarfstd.org/doc/DWARF4.pdf.

[9] pyelftools 0.29 – Library for analyzing ELF files and DWARF debugging information.
https://pypi.org/project/pyelftools/.

[10] AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 1: REQUIRED SERVICES.
https://standards.globalspec.com/std/14214067/arinc-653p1.

[11] AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 0: OVERVIEW OF ARINC 653.
https://standards.globalspec.com/std/13404675/arinc-653p0.

[12] SCons 4.3.0. – a Software Construction Tool.
https://scons.org/doc/production/HTML/scons-user.html.

[13] Introduction to the DWARF Debugging Format.
https://dwarfstd.org/doc/Debugging%20using%20DWARF-2012.pdf.