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

Доледенок Максим Вадимович

Инструмент динамического анализа корректности раскладки памяти в ОСРВ с поддержкой ARINC 653

Курсовая работа

Научный руководитель:
к.ф.-м.н.
Хорошилов Алексей Владимирович

Научный консультант:
Чепцов Виталий Юрьевич

Москва, 2022

Содержание

1 Введение

1.1 ОСРВ

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 ОСРВ JetOS

В данной курсовой работе рассматривается операционная система реального времени JetOS, разрабатываемая в Институте Системного Программирования Российской Академии Наук. ОСРВ JetOS предназначена для использования в авионике, в частности, в гражданских самолетах и спутниках. JetOS разрабатывается в соответствии со стандартом ARINC 653, 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 Статическая конфигурация памяти

В JetOS 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 ОСРВ JetOS детерминированность является очень важным фактором, повышающим безопасность и отказоустойчивость операцион-

ной системы. Поэтому в JetOS применяется строго статическая конфигурация памяти.

1.4 Тестирование раскладки памяти

Поскольку в ОСРВ JetOS требования к безопасности строгие, необходимо проверять, что раскладка памяти происходит корректно. Из-за того что в раскладке памяти много этапов, проверять каждый из них было бы сложно. Поэтому разумным представляется вариант тестирования в динамике, то есть тестирование должно происходить во время исполнения программы операционной системой. Тогда условия тестирования будут максимально близки к реальным условиям использования ОСРВ.

2 Постановка задачи

Целью данной курсовой работы является разработка и интеграция в систему тестирования инструмента для тестирования раскладки памяти в ОСРВ JetOS.

Для достижения цели курсовой работы нужно решить следующие задачи:

- Необходимо изучить структуру раскладки памяти в ОСРВ JetOS.
- Придумать схему динамического тестирования раскладки памяти.
- Реализовать тестирование памяти разделов.
- Реализовать тестирование памяти ядра ОС.
- Реализовать быстрый алгоритм тестирования, который тестирует память выборочно.
- Интегрировать тесты в систему тестирования.

3 Обзор существующих решений

Подобных инструментов в других ОС найти не удалось. Есть общеизвестные инструменты динамического тестирования памяти, например Address Sanitizer. Это инструмент, который обнаруживает различные ошибки по работе с памятью. Он обнаруживает переполнение буфера, использование памяти после её обвобождения, обращения по невалидному адресу и прочие. Но Address Sanitizer не подходит, потому что он проверяет корректность работы с памятью самой программы, а не устройство раскладки памяти.

В самой ОСРВ JetOS есть встроенная проверка раскладчика памяти, но она проверяет конфигурацию памяти статически, без запуска проекта. И проверяется только корректная работа раскладчика. А ошибки могут происходить и на стадии загрузки ОС. Из-за этого невозможно быть полностью уверенным в корректной раскладке памяти. Например, права доступа к памяти при раскладке могут назначаться одни, а в реальности, в результате неправильной работы загрузчика, могут оказаться другими.

4 Исследование и простроение решения задачи

4.1 Устройство памяти в ОСРВ JetOS

Основные принципы устройства памяти в ОСРВ JetOS:

- 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;
- и виртуальная, и физическая память делятся на страницы;
- размеры страниц одинаковы для виртуальной и физической памяти;
- адрес страницы пропорционален ее размеру.

4.2 Блоки памяти

В результате работы раскладчика в виртуальной памяти выделяются блоки памяти. Блок памяти — это непрерывная область виртуальной памяти. Каждый раздел обладает своими блоками памяти, к которым нет доступа у других разделов (если этот блок не является разделяемым). Также у каждого модуля есть свои блоки памяти ядра, к которым нет доступа из разделов и из других модулей.

Блок памяти обладает следующими атрибутами:

- виртуальный адрес блока памяти;
- физический адрес блока памяти;
- размер блока памяти;
- коэффициент выравнивания;
- политика кэширования;
- права доступа: на чтение, на запись, на исполнение;
- наличие зоны безопасности перед блоком и/или после блока;
- размер зон безопасности перед блоком и/или после блока;
- флаг непрерывности блока в физической памяти;
- характер блока памяти: является ли он блоком оперативной памяти или блоком памяти физических устройств;
- список непрерывных отображений виртуальных адресов блока памяти на физические адреса;
- некоторые другие атрибуты.

Ещё блок памяти может быть разделяемым. Это значит, что он доступен сразу нескольким разделам. Каждый раздел имеет доступ к такому блоку с правами доступа, описанными для каждого раздела. Причем разделы могут быть при этом в разных модулях.

Виртуальная память между блоками памяти может иметь либо права доступа одного из двух соседних блоков памяти, либо объединение прав доступа соседних блоков, либо вообще не иметь прав доступа ни на чтение, ни на запись, ни на исполнение.

4.3 Требования к тестированию

Тестирование в основном заключается в проверке прав доступа разделов или ядра к некоторым адресам виртуальной памяти.

Для каждого раздела нужно тестировать права доступа к:

- своим блокам памяти;
- блокам памяти ядра;
- зонам безопасности до/после блоков памяти;
- памяти между блоками;
- разделяемым блокам памяти.

Для ядра нужно тестировать права доступа к:

- своим блокам памяти;
- блокам памяти разделов;
- зонам безопасности до/после блоков памяти;
- памяти между блоками.

Также надо тестировать разделяемые блоки памяти. То есть проверять, что если раздел записал данные в разделяемый блок, то другой раздел может эти данные считать.

4.4 Основные принципы тестирования раскладки памяти

4.4.1 Конфигурация

Программа для тестировния не может находиться в разделе, потому что у раздела ограниченные возможности. Например, из раздела нельзя будет посмотреть файлы конфигурации памяти, которые формирует раскладчик. Значит, инструмент для тестирования должен находиться вне JetOS. Инструмент для тестирования называется memtest, и представляет собой программу на языке Python. Python был выбран из-за того, что система сборки JetOS написана на Python, а также там есть нужные классы для блоков памяти, разделяемых блоков и чтения файлов формата yaml. Именно в этом формате хранятся файлы конфигурации памяти.

Общение с разделами происходит через UART. Для этого в JetOS есть такая абстракция, как потоки, каждый из которых связывается с TCP-портом, к которому может подключиться memtest. Из TCP-порта данные отправляются в UART, из которого

уже их читает JetOS. Для каждого модуля выделяются свои потоки. Основными являются потоки USR и KRN, через USR поток осуществляется ввод/вывод для разделов, а через KRN поток – ввод/вывод для ядра. USR поток является общим для всех разделов.

Чтобы условия тестирования были максимально приближены к реальным, чтение, запись или исполнение памяти осуществляется разделом при тестировании раздела, и ядром при тестировании ядра. Причем если в проекте несколько разделов, нужно тестировать память каждого. Для этого в JetOS была реализована библиотека, которая при тестировании автоматически подключается к разделу и начинает ждать сообщений от memtest. То есть осуществляет чтение в бесконечном цикле.

Поскольку USR поток общий для всех разделов в одном модуле, посылать через него сообщения разным разделам было бы достаточно сложно. Поэтому для тестирования разных разделов использовался механизм расписаний в JetOS. Расписание задаёт порядок и периодичность выполнения разделов. При тестирования раскладки памяти для каждого раздела создаётся своё расписание, в котором всё свободное время процессора выделяется ему. В итоге тестируется сначала один раздел, потом переключается расписание, тестируется другой раздел, и так далее.

4.4.2 Исполнение команд в JetOS при тестировании раздела

Схему тестирования со стороны JetOS можно посмотреть на рис. 1. Разделу посылаются команды на чтение, запись или исполнение данных по определенному виртуальному адресу. Раздел пытается выполнить эту команду. Если команда была выполнена успешно, раздел отвечает инструменту. Иначе происходит прерывание, управление передаётся обработчику прерываний. В нём обработчик передаёт управление обработчику прерываний раздела, такой инструментарий есть в JetOS. Этот обработчик пишет в USR поток сообщение о прерывании и перезапускает раздел. Далее раздел снова ждёт сообщений от инструмента.

4.4.3 Исполнение команд в JetOS при тестировании ядра

Разделу посылаются команды на чтение, запись или исполнение данных по определенному виртуальному адресу. Раздел вызывает системный вызов, в аргументах передавая адрес. Этот вызов поступает обработчику системных вызовов, который находится в библиотеке, подключаемой к ядру при тестировании раскладки памяти. В обработ-

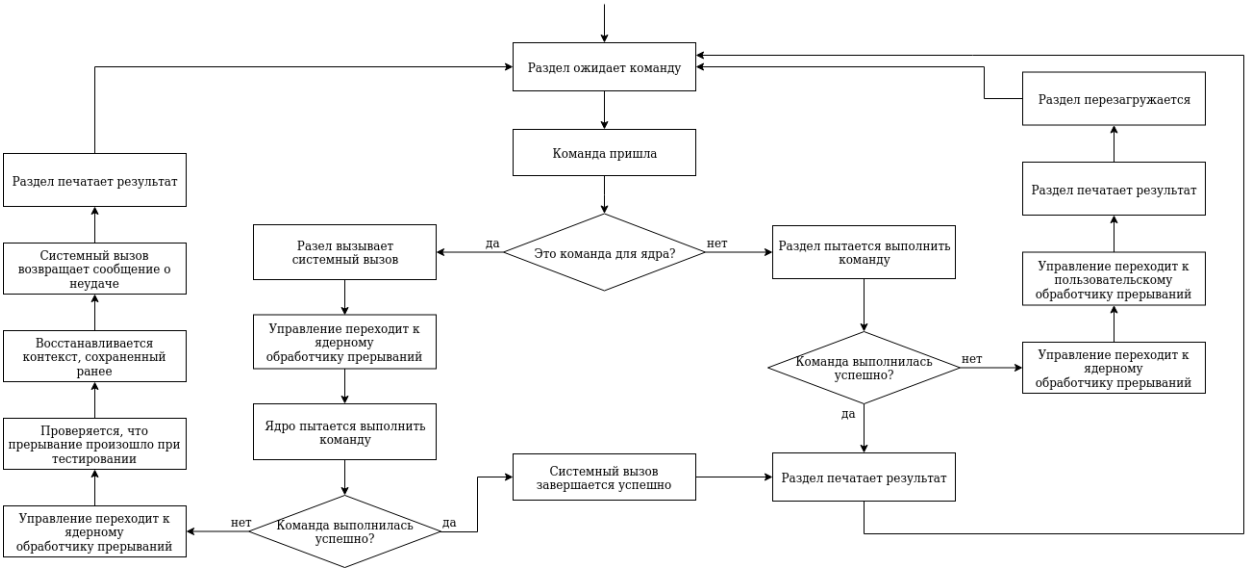

Рис. 1: Схема тестирования со стороны JetOS

чике ядро сначала сохраняет текущий контекст, потом пытается считать, записать или исполнить память по определенному адресу. Если это удаётся, системный вызов завершается успешно, управление возвращается разделу, и он пишет сообщение об успешно выполненной команде. Иначе происходит прерывание, и в его обработчике проверяется, равен ли адрес, при чтении/записи/исполнении которого возникло прерывание, адресу, который был передан в системном вызове. И если равен, то восстанавливается контекст, сохранённый ранее, и системный вызов возвращает сообщение о неудаче. И раздел пишет сообщение о невыполненной команде.

4.5 Протокол взаимодействия memtest с разделом

При запуске memtest посылает проверочное сообщение определённому разделу в определенном модуле, в качестве конкретных чисел выбрано "`0x11 0x88 0xff 0x0`".

Раздел посылает подтверждение вида "`0x11 0x88`". После того, как инструмент получил подтверждение, он может посылать запросы разделу.

Запросы разделу посылаются в виде 4 или более шестнадцатиричных чисел: "`<command> <address> <size> <data>`", где

- `command` – запрос на действие:

- 1 – запрос на выполнение памяти из пользовательского режима;
- 2 – запрос на запись в память из пользовательского режима;
- 3 – запрос на запись с последующим чтением памяти из пользовательского режима;
- 4 – запрос на чтение памяти из пользовательского режима;
- 5 – запрос на переключение расписания модуля;
- 6 – запрос на чтение ядерной памяти из ядра;
- 7 – запрос на запись в ядерную память из ядра;
- 8 – запрос на выполнение ядерной памяти из ядра;
- 9 – запрос на чтение пользовательской памяти из ядра;
- 10 – запрос на запись в пользовательскую память из ядра;
- 11 – запрос на выполнение пользовательской памяти из ядра.

- `address` – адрес от 0 до 2^N - 1, по которому хотим считать, записать или исполнить, где N – размер слова;
- `size` – размер считываемой или записываемой информации в байтах;
- `data` – данные для записи в виде списка байт. Если был запрос на чтение или выполнение, передаётся "0".

Раздел посылает ответы, если удалось выполнить действие, в виде 2 или более шестнадцатичных чисел: "`<command> <data>`", где

- `command` – число от 1 до 11 – сообщение об удачно выполненном запросе (возможные запросы описаны выше);
- `data` – если был запрос на чтение, то передаются считанные данные побайтово. При запросе на запись передаётся информация, считанная из места, куда пытались записать.

Если произошло прерывании при попытке выполнить запрос, раздел отвечает "`0xff <address>`", где `address` – адрес, по которому пытались считать, записать или выполнить. Либо выводится сообщение о прерывании вида

`User code at 0x80001000 address tried to execute unmapped code at 0x803e7000 address`

4.6 Устройство инструмента memtest

Сам инструмент – это программа на языке Python.

Основные этапы его работы:

1. Дождаться запуска JetOS.
2. Сформировать нужные структуры для тестирования из файлов конфигурации, которые ранее создал раскладчик.
3. Протестировать память ядра первого модуля.
4. Отправить команду на переключение расписания, чтобы тестировать нужный раздел в модуле.
5. Протестировать первый раздел первого модуля, вернуться к пункту 4, протестировать второй раздел и т.д.
6. Протестировать разделяемую память первого модуля.
7. Протестировать остальные модули аналогично пунктам 3-6.
8. Протестировать разделяемую память, общую для нескольких модулей.
9. Напечатать результаты тестирования.

Само тестирование памяти для каждого раздела и для ядра происходит последовательно, начиная с минимального виртуального адреса, и заканчивая максимальным. При этом, тестировать каждый виртуальный адрес было бы слишком долго, поэтому тестируется один виртуальный адрес из страницы.

4.7 Алгоритм быстрого тестирования

Но даже тестирование по одному виртуальному адресу из каждой страницы занимает достаточно много времени. Поэтому был реализован алгоритм быстрого тестирования. Этот алгоритм для каждой непрерывной области памяти с одинаковыми правами

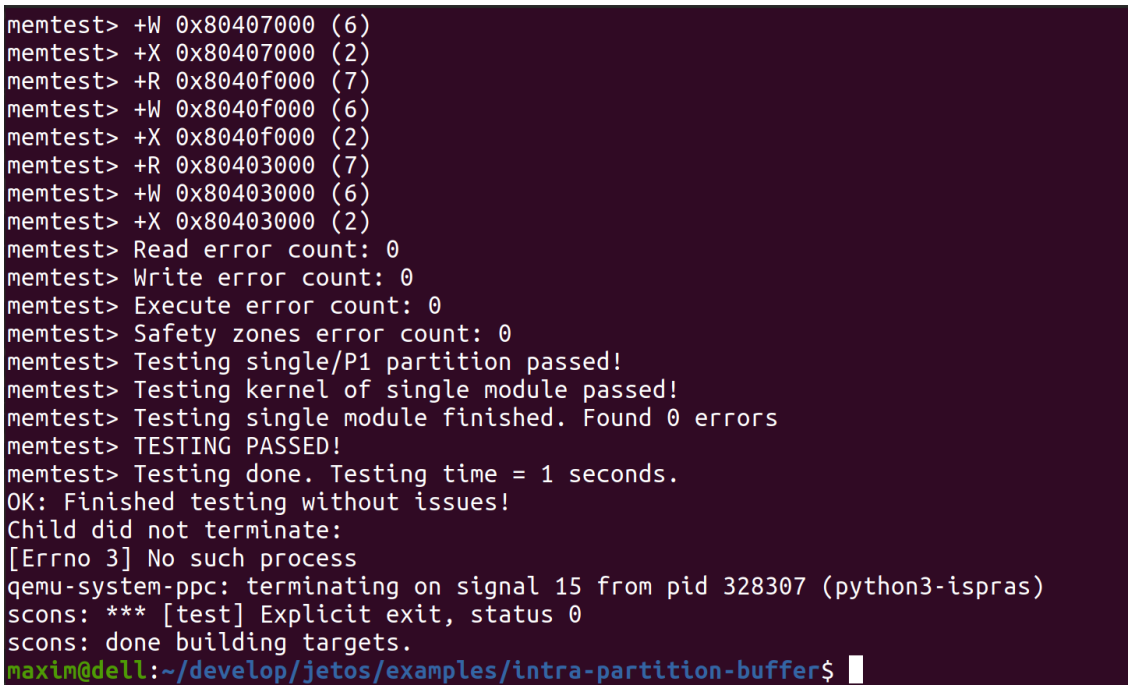

Рис. 2: пример трассы при завершении тестирования без ошибок

тестирует только 2 страницы. Такими областями с одинаковыми правами являются например блок памяти или зона безопасности. И если основной алгоритм тестирования может работать десятки часов, то быстрый алгоритм работает чаще всего не больше минуты, при этом позволяя выявлять основные ошибки, допущенные при раскладке памяти.

4.8 Запуск тестирования

Сейчас инструмент memtest внедрён в систему тестирования JetOS. Чтобы протестировать какой-нибудь проект, нужно в его главной директории запустить команду

```
$ isp-test memtest=1
```

И в случае положительного исхода тестирования, вывод будет как на рис. 2.

Если же на каком-то этапе тестирования произойдет ошибка, будет выдано сообщение об ошибке как на рис. 3.

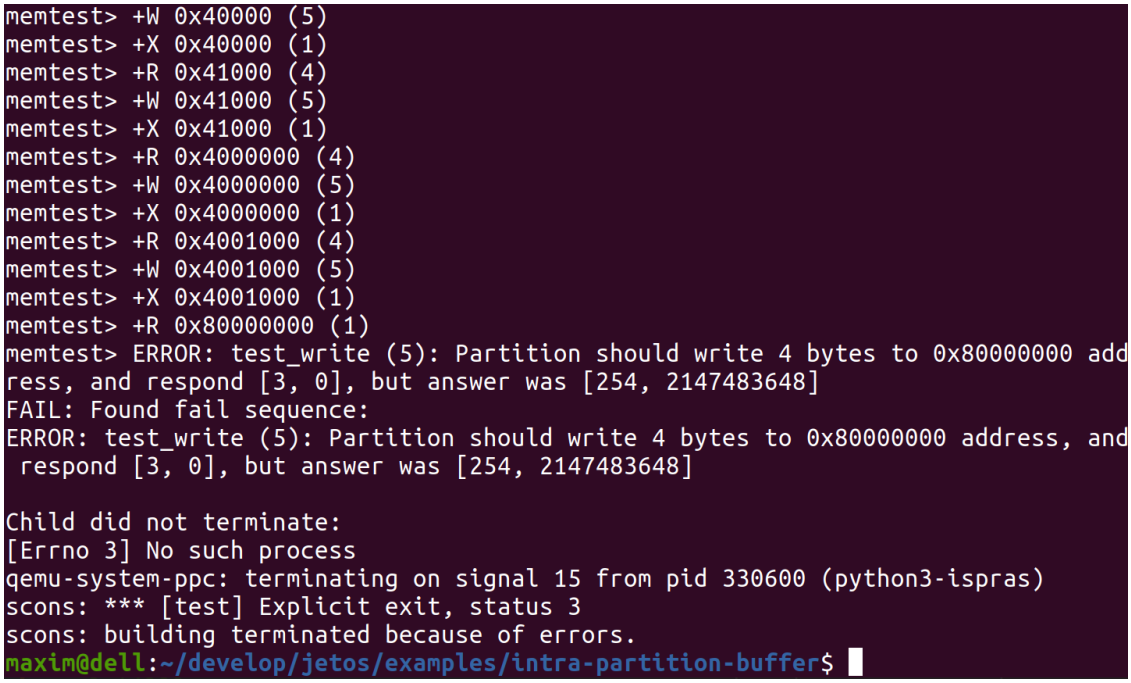

Рис. 3: пример трассы при завершении тестирования с ошибкой

4.9 Результаты тестирования

В результате быстрого тестирования на разных архитектурах были выявлены ошибки:

1. На архитектуре MIPS раздел имел права доступа 'RWX' для блоков ядра. На уровне загрузчика не было никакой защиты ядерных блоков.

2. На архитектуре PowerPC ядро могло читать данные из пользовательских блоков с правами только на чтение. Ошибка была на стадии создания конфигурации для раскладчика ОС.

3. На архитектуре x86 раздел мог записывать данные в блок .ELF.0 с правами 'RX'. В раскладчике памяти были неверно указаны права доступа к пользовательской RO (только на чтение) памяти.

4. При попытке чтения/записи ядром ОС в блок .LOADER_SPIN на проектах, использующих одно ядро, возникало прерывание, хотя там права 'RW'. На самом

деле, для проектов с одним ядром, этот блок памяти не должен выделяться совсем, но он выделялся на стадии загрузки ОС.

5. На архитектуре PowerPC ядро могло исполненять данные в блоке памяти .LOADER_FS, хотя там стоят права 'R'. Неверные права выставлялись в загрузчике.

Также в процессе тестирования было найдено несколько ошибок, непосредственно не связанных с ошибками при раскладке памяти. Все эти ошибки уже исправлены.

5 Заключение

В рамках курсовой работы была изучена работа раскладчика в ОСРВ JetOS, спроектирован и реализован инструмент для тестирования раскладки памяти. Он был интегрирован в систему тестирования, и сейчас быстрое тестирование используется в автоматических тестах ОСРВ JetOS. В результате быстрого тестирования было найдено 5 ошибок раскладки памяти, которые показывают, что реализация праактической части курсовой работает исправно.

6 Дальнейшие планы

- Написать для memtest особенные проекты для тестирования, в которых структура памяти будет сложнее, чтобы более полно проверить корректность раскладки памяти. Сейчас тестирование происходит на уже существующих проектах для JetOS.

- При полном тестировании в зависимости от архитектуры можно существенно сократить количество обращений к памяти из-за того, что память выделяется блоками фиксированного размера. Также, поскольку полное тестирование памяти выполняется достаточно долго, можно по ходу тестирования выдавать уверенность в корректном результате. Причем чтение из разных адресов будет повышать эту уверенность по-разному, потому что виртуальный адрес, по которому раскладчик выделил блок памяти, кратен размеру этого блока. Из-за этого проверка корректности раскладки по адресам, имеющим бóльшую кратность размеру страницы, в теории может отловить больше ошибок раскладчика, чем проверка по адресам, имеющим меньшую кратность.

- Сделать стратегию быстрого тестирования оптимальной, теоретически обосновать её оптимальность, опираясь на выводы из предыдущего пункта.

Список литературы

[1] ARINC Avionics Standards // SAE ITC [Электронный ресурс]. — URL: https://www.aviation-ia.com/product-categories/600-series/ (дата обращения: 30.05.2022).

[2] Address Sanitizer // Clang Team [Электронный ресурс]. — URL: https://clang.llvm.org/docs/AddressSanitizer.html (дата обращения: 30.05.2022).

[3] SCons 4.3.0 // The SCons Foundation [Электронный ресурс]. — URL: https://scons.org/doc/production/HTML/scons-user.html (дата обращения: 30.05.2022).

[4] JetOS // Redmine [Электронный ресурс]. — URL: https://forge.ispras.ru/projects/jetos/repository/main/revisions/master/entry/README.md (дата обращения: 30.05.2022).

[5] JetOS allocator // Redmine [Электронный ресурс]. — URL: https://forge.ispras.ru/projects/jetos/repository/allocator/revisions/master/entry/README.txt (дата обращения: 30.05.2022).