Чекановкин А.Е. Архитектура многопоточного приложения в условиях интероперабельности процессов // Международный журнал информационных технологий и энергоэффективности. – 2023. – Т. 8 № 6(32) с. 40–49

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004

АРХИТЕКТУРА МНОГОПОТОЧНОГО ПРИЛОЖЕНИЯ В УСЛОВИЯХ ИНТЕРОПЕРАБЕЛЬНОСТИ ПРОЦЕССОВ

Чекановкин А.Е.
ФГБУО ВО «МИРЭА - Российский технологический университет», Москва, Россия, (119454, г. Москва, просп. Вернадского, 78, стр. 4.), e-mail: 4565855@mail.ru

В современных информационных системах все больше возникает необходимость в многопоточных приложениях, способных эффективно работать в условиях, когда процессы взаимодействуют на разных платформах и в различных средах. В таких сценариях интероперабельность процессов становится ключевым фактором для обеспечения правильной работы приложения. В данной статье будет представлена архитектура многопоточного приложения, которая позволяет решить данную проблему и обеспечить эффективное взаимодействие между процессами.

Ключевые слова: многопоточность; гонка данных; интероперабельность процессов.

ARCHITECTURE OF A MULTI-THREAD APPLICATION UNDER THE CONDITIONS OF ARCH PROCESS INTEROPERABILITY

Chekanovkin A.E.
MIREA - Russian Technological University, Moscow, Russia (119454, Moscow, avenue. Vernadsky, 78, b. 4), e-mail: 4565855@mail.ru

In modern information systems, there is an increasing need for multithreaded applications that can work effectively in conditions when processes interact on different platforms and in different environments. In such scenarios, the interoperability of processes becomes a key factor to ensure the correct operation of the application. This article will present the architecture of a multithreaded application, which allows you to solve this problem and ensure effective interaction between processes.

Keywords: multithreading; data race; process interoperability.

Определение многопоточности

Многопоточность - это концепция в программировании, которая позволяет одновременно выполнять несколько потоков внутри одного процесса. Поток представляет собой независимую единицу выполнения, которая может выполняться параллельно или конкурентно с другими потоками. Каждый поток имеет собственный стек вызовов, указатель команд и состояние выполнения [1].

Основная идея многопоточности заключается в том, чтобы разделить задачу на более мелкие подзадачи и выполнять их параллельно в разных потоках. Это позволяет повысить эффективность и производительность программы, так как различные потоки могут выполнять вычисления, ввод-вывод операции или другие операции одновременно.

В отличие от выполнения в однопоточных приложениях, многопоточные приложения могут более эффективно использовать доступные ресурсы системы, такие как процессорное время и память. Кроме того, многопоточность позволяет реагировать на асинхронные события и организовывать параллельное выполнение различных задач.

Преимущества многопоточных приложений включают улучшенную отзывчивость, более эффективное использование ресурсов, улучшенную масштабируемость и возможность реализации сложных задач, которые требуют параллельного выполнения.

Однако, при разработке многопоточных приложений также возникают ряд сложностей и проблем. Например, необходимо управлять синхронизацией доступа к общим ресурсам, чтобы избежать состояний гонки и других проблем совместного доступа к данным. Кроме того, необходимо тщательно планировать и координировать выполнение потоков, чтобы избежать блокировок и конфликтов. Эти проблемы требуют особого внимания при проектировании и разработке многопоточных приложений.

Преимущества многопоточных приложений

Многопоточные приложения предлагают ряд значительных преимуществ, которые делают их предпочтительным выбором при разработке современных программных систем. Вот некоторые из основных преимуществ многопоточных приложений [2]:

Повышенная производительность: Многопоточные приложения позволяют использовать параллельное выполнение задач на нескольких процессорных ядрах. Это может привести к существенному увеличению производительности и сокращению времени выполнения программы. Разделение работы на небольшие задачи, которые выполняются параллельно, позволяет более эффективно использовать вычислительные ресурсы системы.

Улучшенная отзывчивость: Многопоточность позволяет отделить длительные операции, такие как сетевые запросы или обработка пользовательского ввода, от основного потока выполнения приложения. Это позволяет приложению оставаться отзывчивым на взаимодействие с пользователем, даже если происходят другие вычислительные операции.

Лучшее использование ресурсов: Многопоточные приложения позволяют более эффективно использовать доступные ресурсы системы, такие как процессорное время и память. При наличии нескольких потоков, связанных с различными задачами, можно более равномерно распределить вычислительную нагрузку и избежать простоев, которые могут возникнуть при выполнении операций последовательно.

Возможность параллельной обработки данных: В многопоточных приложениях можно одновременно обрабатывать различные наборы данных или выполнять вычисления на разных частях данных. Это особенно полезно при работе с большими объемами данных, такими как обработка видео, анализ больших наборов данных или распараллеливание сложных вычислений.

Легкость разработки и поддержки: Многопоточные приложения позволяют разделить задачи на более мелкие и независимые единицы, что упрощает их разработку и поддержку. Каждый поток может быть разработан и отлажен отдельно, что упрощает их тестирование и обнаружение ошибок. Кроме того, многопоточные приложения могут быть более гибкими и масштабируемыми, так как дополнительные потоки могут быть добавлены или удалены в зависимости от требований системы.

В целом, многопоточные приложения предоставляют мощный инструмент для эффективной организации и управления выполнением задач. Они способствуют повышению производительности, улучшению отзывчивости и более эффективному использованию ресурсов системы, что делает их незаменимыми во многих современных программных системах.

Проблемы при разработке многопоточных приложений

Разработка многопоточных приложений может столкнуться с рядом сложностей и проблем, связанных с эффективностью, безопасностью и корректностью выполнения [3]. Некоторые из основных проблем, с которыми разработчики могут столкнуться при создании многопоточных приложений, включают:

Состояние гонки (Race conditions): Состояние гонки возникает, когда несколько потоков одновременно обращаются к общему ресурсу и пытаются изменить его состояние. Это может привести к непредсказуемым результатам или ошибкам в программе. Разработчики должны применять механизмы синхронизации, такие как мьютексы или блокировки, для предотвращения состояний гонки и обеспечения правильного доступа к общим ресурсам.

Взаимные блокировки (Deadlocks): Взаимная блокировка возникает, когда два или более потока ожидают освобождения ресурсов, заблокированных другими потоками. Это может привести к застою и невозможности продолжить выполнение программы. Разработчики должны аккуратно планировать использование блокировок и управлять порядком их получения, чтобы избежать взаимных блокировок.

Голодание потоков (Thread starvation): Голодание потоков возникает, когда один или несколько потоков систематически не получают достаточно ресурсов для выполнения своих задач. Это может привести к снижению производительности и отзывчивости приложения. Разработчики должны учитывать приоритеты потоков, правильно управлять ресурсами и предотвращать голодание потоков.

Неправильное использование синхронизации: Неправильное использование механизмов синхронизации, таких как блокировки или семафоры, может привести к непредсказуемому поведению приложения или проблемам с производительностью. Разработчики должны тщательно планировать и применять синхронизацию, чтобы избежать ненужной блокировки и улучшить производительность.

Управление потоками и планирование: Управление потоками и планирование их выполнения может быть сложной задачей, особенно при наличии большого количества потоков. Неправильное планирование или некорректное использование ресурсов процессора может привести к снижению производительности или неравномерному использованию вычислительных ресурсов.

Отладка и тестирование: Отладка многопоточных приложений может быть сложной из-за непредсказуемого поведения и взаимодействия между потоками. Тестирование таких приложений также требует особого внимания, чтобы убедиться в их корректности и надежности при параллельном выполнении.

Все эти проблемы требуют тщательного планирования, проектирования и тестирования многопоточных приложений. Разработчики должны иметь хорошее понимание принципов синхронизации и механизмов управления потоками, чтобы избежать потенциальных проблем и обеспечить корректное функционирование многопоточных приложений.

Определение интероперабельности процессов

Интероперабельность процессов - это способность различных процессов или программных систем взаимодействовать и совместно работать друг с другом, несмотря на различия в их архитектуре, платформе или языке программирования [4]. Интероперабельность процессов является ключевым аспектом в современных информационных системах, где различные компоненты или приложения часто должны обмениваться данными и выполнять совместные операции.

Интероперабельность процессов предполагает, что процессы должны быть в состоянии взаимодействовать, обмениваться информацией и использовать функциональность друг друга без необходимости полной переработки или изменения существующих компонентов. Она позволяет различным процессам работать в согласованном и совместимом окружении, даже если они были разработаны независимо друг от друга или используют разные технологии.

Одной из основных целей интероперабельности процессов является создание гибкого и расширяемого программного обеспечения, которое может взаимодействовать с другими системами и использовать их возможности. Она способствует повторному использованию кода, ускоряет разработку и позволяет строить более сложные системы, объединяющие функциональность различных компонентов и приложений.

Интероперабельность процессов может быть достигнута через использование стандартов и протоколов обмена данными, таких как XML, JSON, REST или SOAP. Эти стандарты обеспечивают единый способ представления и передачи данных между процессами, независимо от их платформы или языка программирования.

Подходы к обеспечению интероперабельности процессов могут варьироваться в зависимости от конкретных потребностей системы. Они могут включать разработку адаптеров или посредников, которые обеспечивают преобразование данных между различными форматами или обеспечивают согласованность взаимодействия между процессами. Также может использоваться разработка API или сервис-ориентированная архитектура (SOA) для предоставления стандартизированных интерфейсов, через которые процессы могут взаимодействовать друг с другом.

В итоге, интероперабельность процессов играет важную роль в построении сложных и совместных систем. Она обеспечивает возможность интеграции различных компонентов и приложений, улучшает гибкость и эффективность системы, а также способствует повышению производительности и функциональности информационных систем.

Проблемы, связанные с интероперабельностью процессов

В процессе обеспечения интероперабельности процессов могут возникать различные проблемы, которые затрудняют успешное взаимодействие и совместную работу различных компонентов или приложений. Некоторые из основных проблем, связанных с интероперабельностью процессов, включают:

Различия в форматах данных и протоколах: Различные процессы или системы могут использовать разные форматы данных и протоколы обмена информацией. Например, одна система может использовать формат XML, в то время как другая - JSON. Это может привести к сложностям при передаче и преобразовании данных между различными форматами. Разработчики должны быть готовы к реализации механизмов преобразования данных и обеспечению согласованности форматов при взаимодействии между процессами.

Различия в языках программирования и платформах: Различные процессы могут быть написаны на разных языках программирования или работать на разных платформах. Это может вызвать проблемы совместимости и взаимодействия. Разработчики должны искать решения, такие как использование промежуточного программного интерфейса (API) или разработку адаптеров, чтобы обеспечить возможность взаимодействия между процессами, независимо от их языка или платформы.

Сложность обеспечения безопасности: Интероперабельность процессов может представлять риски с точки зрения безопасности. При взаимодействии между различными процессами могут возникать уязвимости или возможность несанкционированного доступа к данным. Разработчики должны обеспечивать надежность и защищенность взаимодействия, используя механизмы аутентификации, шифрования и контроля доступа.

Управление версиями и изменениями: При взаимодействии между процессами могут возникать проблемы с управлением версиями и изменениями в интерфейсах и протоколах. Обновление или изменение одного процесса может потребовать соответствующих изменений в других процессах, чтобы сохранить совместимость. Разработчики должны учитывать вопросы управления версиями и разрабатывать стратегии обновления и поддержки интерфейсов для обеспечения согласованности и продолжительности работы системы.

Отладка и тестирование: Интероперабельность процессов усложняет процесс отладки и тестирования системы в целом. При возникновении проблем взаимодействия может быть сложно определить и исправить причину. Тестирование должно включать проверку взаимодействия между различными компонентами и проверку соответствия спецификациям и стандартам интероперабельности.

Разработчики многопоточных приложений должны быть готовы к решению этих проблем и применению соответствующих методов и технологий для обеспечения успешной интероперабельности процессов. Это требует внимательного планирования, проектирования и тестирования, а также учета стандартов и протоколов, чтобы обеспечить гармоничное взаимодействие и эффективную работу системы.

Требования к архитектуре многопоточного приложения в условиях интероперабельности процессов

Архитектура многопоточного приложения должна быть разработана с учетом требований интероперабельности процессов, чтобы обеспечить эффективное взаимодействие и совместную работу различных компонентов или приложений. Ключевые требования, которые следует учитывать при проектировании архитектуры:

Стандартизированные протоколы и форматы данных: При проектировании архитектуры необходимо использовать стандартизированные протоколы и форматы данных для обмена информацией между процессами. Это обеспечит совместимость и возможность взаимодействия даже в случае различных языков программирования или платформ.

Гибкость и расширяемость: Архитектура должна быть гибкой и расширяемой, чтобы легко включать новые компоненты или приложения в систему. Это позволит легко интегрировать новые функциональные возможности и обеспечивать совместную работу с другими процессами.

Стабильность и надежность: Архитектура должна быть стабильной и надежной для обеспечения непрерывной работы системы. Взаимодействие между процессами должно быть надежным и обеспечивать сохранность данных и целостность операций.

Управление версиями и изменениями: Архитектура должна предусматривать механизмы управления версиями и изменениями в интерфейсах и протоколах. Это позволит гибко вносить изменения и обновления в систему без нарушения совместимости и взаимодействия с другими процессами.

Безопасность: Безопасность является важным аспектом интероперабельности процессов. Архитектура должна обеспечивать механизмы аутентификации, авторизации и шифрования для защиты данных и обеспечения безопасного взаимодействия между процессами.

Масштабируемость: Архитектура должна быть масштабируемой, чтобы обеспечить эффективную работу системы при увеличении нагрузки или добавлении новых процессов. Это включает возможность горизонтального и вертикального масштабирования, а также управление ресурсами.

Четкая документация и соглашения: Важно иметь четкую документацию и соглашения относительно интерфейсов, протоколов и общих правил взаимодействия между процессами. Это поможет разработчикам понять и правильно использовать интерфейсы и обеспечит согласованность взаимодействия.

Учитывая эти требования при проектировании архитектуры многопоточного приложения, можно создать систему, которая обеспечивает эффективное взаимодействие и совместную работу различных процессов или приложений, даже при различных языках программирования, платформах или стандартах.

Компоненты архитектуры

Архитектура многопоточного приложения, ориентированного на интероперабельность процессов, состоит из различных компонентов, которые совместно работают для обеспечения эффективного взаимодействия и совместной работы. Вот некоторые основные компоненты, которые можно встретить в такой архитектуре [5]:

Потоки (Threads): Потоки представляют собой базовые строительные блоки многопоточного приложения. Они представляют собой независимые исполняющиеся единицы, которые могут параллельно выполнять различные задачи. Каждый процесс может содержать несколько потоков, которые совместно работают для выполнения общей задачи.

Межпроцессное взаимодействие (Interprocess Communication, IPC): Для обмена данными и синхронизации работы между процессами необходимо использовать механизмы межпроцессного взаимодействия. Это может включать различные методы, такие как использование сокетов, пайпов, разделяемой памяти или сообщений. Механизмы IPC позволяют процессам обмениваться информацией и координировать свою работу.

Промежуточные компоненты (Middleware): Промежуточные компоненты представляют собой слой абстракции, который облегчает взаимодействие между различными процессами и обеспечивает уровень интероперабельности. Эти компоненты могут включать в себя различные технологии, такие как сервисы сообщений, шины данных, адаптеры протоколов и т. д. Промежуточные компоненты упрощают взаимодействие между процессами, скрывая детали реализации и обеспечивая единый интерфейс.

Сервисы (Services): Сервисы представляют собой компоненты, которые предоставляют определенные функциональные возможности для других процессов или приложений. Они могут включать в себя сервисы авторизации, аутентификации, управления данными и другие сервисы, необходимые для эффективной работы системы. Сервисы могут быть предоставлены как локально внутри одного процесса, так и удаленно через механизмы удаленного вызова процедур (Remote Procedure Call, RPC) или веб-сервисы.

Интерфейсы (Interfaces): Интерфейсы определяют способ взаимодействия между компонентами и процессами. Они определяют набор методов, функций или протоколов, которые могут быть использованы для передачи данных и выполнения операций. Интерфейсы должны быть ясно задокументированы и соответствовать стандартам, чтобы обеспечить совместимость и возможность взаимодействия между компонентами.

Компоненты архитектуры многопоточного приложения должны быть тщательно спроектированы и интегрированы, чтобы обеспечить эффективное взаимодействие и совместную работу процессов в условиях интероперабельности. Это требует понимания требований системы, выбора подходящих технологий и стандартов, а также грамотной реализации и интеграции компонентов.

Организация взаимодействия между потоками

В многопоточных приложениях, ориентированных на интероперабельность процессов, взаимодействие между потоками играет важную роль для эффективной работы и совместного выполнения задач. Рассмотрим несколько распространенных методов организации взаимодействия между потоками [6]:

Синхронизация (Synchronization): Синхронизация является ключевым аспектом взаимодействия между потоками. Она позволяет координировать выполнение задач и обеспечивает согласованность данных. Механизмы синхронизации, такие как блокировки, семафоры, мьютексы и условные переменные, используются для предотвращения состояния гонки и обеспечения доступа к общим ресурсам.

Потокобезопасные структуры данных (Thread-Safe Data Structures): Потокобезопасные структуры данных предназначены для обеспечения безопасного доступа к общим данным из разных потоков. Это могут быть потокобезопасные очереди, хеш-таблицы, списки и другие структуры данных, которые обеспечивают синхронизированный доступ к данным и предотвращают конфликты.

Каналы и очереди (Channels and Queues): Использование каналов и очередей позволяет организовать асинхронное взаимодействие между потоками. Один поток может помещать данные в очередь или канал, а другой поток может извлекать их и обрабатывать. Это обеспечивает асинхронность и гибкость взаимодействия между потоками.

Использование событий и сигналов (Events and Signals): События и сигналы используются для уведомления потоков о наступлении определенных событий или изменениях состояния. Они позволяют потокам ожидать определенных условий и реагировать на них. Это может быть полезно при ожидании завершения операции или сигнализации о готовности ресурса.

Пулы потоков (Thread Pools): Пулы потоков представляют собой набор предварительно созданных потоков, которые могут использоваться для выполнения задач. Они позволяют эффективно использовать ресурсы и снизить накладные расходы на создание и уничтожение

потоков. Пулы потоков обеспечивают распределение задач и масштабируемость взаимодействия между потоками.

Механизмы обмена сообщениями (Message Passing): Механизмы обмена сообщениями позволяют потокам передавать данные и команды друг другу через сообщения. Это может включать использование очередей сообщений, асинхронных вызовов процедур или других протоколов обмена сообщениями. Механизмы обмена сообщениями обеспечивают асинхронное и гибкое взаимодействие между потоками.

При организации взаимодействия между потоками важно учитывать требования системы и выбрать подходящие механизмы синхронизации и коммуникации. Это поможет обеспечить эффективное и безопасное взаимодействие между потоками, минимизировать состояния гонки и конфликты данных, а также повысить производительность и отзывчивость приложения.

Механизмы синхронизации и координации

Механизмы синхронизации и координации играют важную роль в архитектуре многопоточных приложений, ориентированных на интероперабельность процессов. Они обеспечивают правильное взаимодействие и согласованную работу потоков и процессов. Ниже приведены распространенные механизмы синхронизации и координации [7]:

Блокировки (Locks): Блокировки используются для обеспечения эксклюзивного доступа к ресурсам или критическим секциям кода. Когда поток получает блокировку, другие потоки должны ждать освобождения блокировки, прежде чем получить доступ к защищенным ресурсам. Блокировки могут быть реализованы с использованием мьютексов, семафоров или атомарных операций.

Семафоры (Semaphores): Семафоры позволяют ограничить количество потоков, которые могут одновременно получить доступ к определенному ресурсу или критической секции. Семафор содержит счетчик, который уменьшается при каждом получении доступа потоком и увеличивается при его освобождении. Это позволяет управлять параллельным выполнением задач и предотвращать перегрузку ресурсов.

Условные переменные (Condition Variables): Условные переменные позволяют потокам ожидать определенного условия или события перед продолжением выполнения. Они могут быть использованы для ожидания завершения операции, появления новых данных или изменения состояния. Условные переменные работают совместно с блокировками и позволяют потокам эффективно управлять своим состоянием ожидания.

Синхронизаторы (Synchronizers): Синхронизаторы представляют собой более высокоуровневые абстракции для управления потоками и их взаимодействия. Они включают в себя семафоры, барьеры, счетчики и другие механизмы, которые позволяют потокам синхронизироваться, согласовывать свою работу и добиться определенных условий перед продолжением выполнения.

Координаторы (Coordinators): Координаторы представляют собой компоненты, ответственные за координацию работы потоков и процессов в многопоточных приложениях. Они могут включать в себя централизованные или распределенные механизмы управления, планирования задач, распределения ресурсов и обмена информацией между потоками.

При выборе механизмов синхронизации и координации необходимо учитывать требования приложения, его архитектуру и особенности интероперабельности процессов.

Оптимальный выбор механизмов позволяет избежать состояний гонки, конфликтов и улучшить производительность приложения.

Заключение

В данной статье были представлены механизмы архитектуры многопоточного приложения, обеспечивающие эффективное взаимодействие между потоками в условиях интероперабельности процессов. Описаны основные компоненты архитектуры, механизмы синхронизации и координации. Данная статья может быть полезна для разработчиков, сталкивающихся с необходимостью создания многопоточных приложений, работающих в различных средах и на разных платформах.

Список литературы

1. Understanding Basic Multithreading Concepts [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032e/index.html (дата обращения: 28.05.2023)
2. Фешина, Е. В. Многопоточность и асинхронность в языке программирования Python / Е. В. Фешина, Д. А. Омельченко, Р. Г. Гонатаев // Инновации. Наука. Образование. – 2021. – № 28. – С. 988-992. – EDN ANHVUX.
3. Гладышев Е.И., Мурыгин А.В. Многопоточность в приложениях // Актуальные проблемы авиации и космонавтики. 2012. №8. URL: https://cyberleninka.ru/article/n/mnogopotochnost-v-prilozheniyah (дата обращения: 31.05.2023)
4. Некрасов, А. Г. Проактивный мониторинг процессов интероперабельности и организационной устойчивости в сфере транспорта / А. Г. Некрасов, А. С. Синицына // Академик Владимир Николаевич Образцов - основоположник транспортной науки : труды международной научно-практической конференции, посвященной 125-летию университета, Москва, 22 октября 2021 года. – Москва: Российский университет транспорта, 2021. – С. 637-647. – DOI 10.47581/2022/Obrazcov.85. – EDN ULSRLK.
5. Шалагин, В. И. Основы организации многопоточной обработки данных в вычислительных системах / В. И. Шалагин, А. Л. Марухленко // Современные информационные технологии и информационная безопасность: Сборник научных статей Всероссийской научно-технической конференции, Курск, 17 мая 2022 года. – Курск: Юго-Западный государственный университет, 2022. – С. 40-58. – EDN BCHZOX.
6. Елизаров, В. В. Разработка веб-приложения для организации взаимодействия между пользователями посредством потокового мультимедиа / В. В. Елизаров // NEW SCIENCE GENERATION: сборник статей II Международной научно-практической конференции, Петрозаводск, 25 декабря 2019 года. – Петрозаводск: Международный центр научного партнерства «Новая Наука» (ИП Ивановская Ирина Игоревна), 2019. – С. 239-243. – EDN ZUFRDZ.
7. Методы синхронизации потоков в многопоточном приложении на языке C++ [Электронный ресурс]. – Режим доступа: https://science-pedagogy.ru/ru/article/view?id=1962 (Дата обращения: 29.05.2023).

References

1. Understanding Basic Multithreading Concepts [Elektronnyj resurs]. – Rezhim dostupa: https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032e/index.html (data obrashcheniya: 28.05.2023)
2. Feshina, E. V. Mnogopotochnost' i asinhronnost' v yazyke programmirovaniya Python / E. V. Feshina, D. A. Omel'chenko, R. G. Gonataev // Innovacii. Nauka. Obrazovanie. – 2021. – № 28. – pp. 988-992. – EDN ANHVUX.
3. Gladyshev E.I., Murygin A.V. Mnogopotochnost' v prilozheniyah // Aktual'nye problemy aviacii i kosmonavtiki. 2012. №8. URL: https://cyberleninka.ru/article/n/mnogopotochnost-v-prilozheniyah (data obrashcheniya: 31.05.2023)
4. Nekrasov, A. G. Proaktivnyj monitoring processov interoperabel'nosti i organizacionnoj ustojchivosti v sfere transporta / A. G. Nekrasov, A. S. Sinicyna // Akademik Vladimir Nikolaevich Obrazcov - osnovopolozhnik transportnoj nauki : trudy mezhdunarodnoj nauchno-prakticheskoj konferencii, posvyashchennoj 125-letiyu universiteta, Moskva, 22 oktyabrya 2021 goda. – Moskva: Rossijskij universitet transporta, 2021. – pp. 637-647. – DOI 10.47581/2022/Obrazcov.85. – EDN ULSRLK.
5. SHalagin, V. I. Osnovy organizacii mnogopotochnoj obrabotki dannyh v vychislitel'nyh sistemah / V. I. SHalagin, A. L. Maruhlenko // Sovremennye informacionnye tekhnologii i informacionnaya bezopasnost': Sbornik nauchnyh statej Vserossijskoj nauchno-tekhnicheskoj konferencii, Kursk, 17 maya 2022 goda. – Kursk: YUgo-Zapadnyj gosudarstvennyj universitet, 2022. – pp. 40-58. – EDN BCHZOX.
6. Elizarov, V. V. Razrabotka veb-prilozheniya dlya organizacii vzaimodejstviya mezhdu pol'zovatelyami posredstvom potokovogo mul'timedia / V. V. Elizarov // NEW SCIENCE GENERATION: sbornik statej II Mezhdunarodnoj nauchno-prakticheskoj konferencii, Petrozavodsk, 25 dekabrya 2019 goda. – Petrozavodsk: Mezhdunarodnyj centr nauchnogo partnerstva «Novaya Nauka» (IP Ivanovskaya Irina Igorevna), 2019. – pp. 239-243. – EDN ZUFRDZ.
7. Metody sinhronizacii potokov v mnogopotochnom prilozhenii na yazyke C++ [Elektronnyj resurs]. – Rezhim dostupa: https://science-pedagogy.ru/ru/article/view?id=1962 (Data obrashcheniya: 29.05.2023).