РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 429 526** (13) **C2**

(51) МПК
*G06F 12/00* (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ,
ПАТЕНТАМ И ТОВАРНЫМ ЗНАКАМ

# (12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21)(22) Заявка: **2008116715/08, 16.10.2006**

(24) Дата начала отсчета срока действия патента:
**16.10.2006**

Приоритет(ы):
(30) Конвенционный приоритет:
**26.10.2005 US 60/730,546**
**30.06.2006 US 11/428,162**

(43) Дата публикации заявки: **27.10.2009** Бюл. № 30

(45) Опубликовано: **20.09.2011** Бюл. № **26**

(56) Список документов, цитированных в отчете о поиске: **US 5991518 A, 23.11.1999. US 2002/0099954 A1, 25.07.2002. WO 03038599 A2, 08.05.2003. RU 2004107492 A, 27.09.2005.**

(85) Дата начала рассмотрения заявки PCT на национальной фазе: **25.04.2008**

(86) Заявка PCT:
**US 2006/040527 (16.10.2006)**

(87) Публикация заявки PCT:
**WO 2007/050363 (03.05.2007)**

Адрес для переписки:
**129090, Москва, ул.Б.Спасская, 25, стр.3, ООО "Юридическая фирма Городисский и Партнеры", пат.пов. А.В.Мищу, рег.№ 364**

(72) Автор(ы):
**ХАНТ Гален К. (US),**
**ЛАРУС Джеймс Р. (US),**
**АБАДИ Мартин (US),**
**АЙКЕН Марк (US),**
**БАРХЭМ Пол (US),**
**ФАНДРИЧ Мануэл А. (US),**
**ХОБЛИТЦЕЛ Крис (US),**
**ХОДСОН Орион (US),**
**ЛЕВИ Стивен (US),**
**МЕРФИ Ник (US),**
**СТЕНСГОР Бьярне (US),**
**ТАРДИТИ Дэвид (US),**
**УОББЕР Тэд (US),**
**ЗИЛЛ Брайан (US)**

(73) Патентообладатель(и):
**МАЙКРОСОФТ КОРПОРЕЙШН (US)**

## (54) СТАТИЧЕСКИ ПРОВЕРЯЕМЫЕ ДОПУСКАЮЩИЕ МЕЖПРОЦЕССНЫЙ ОБМЕН ИЗОЛИРОВАННЫЕ ПРОЦЕССЫ

(57) Реферат:

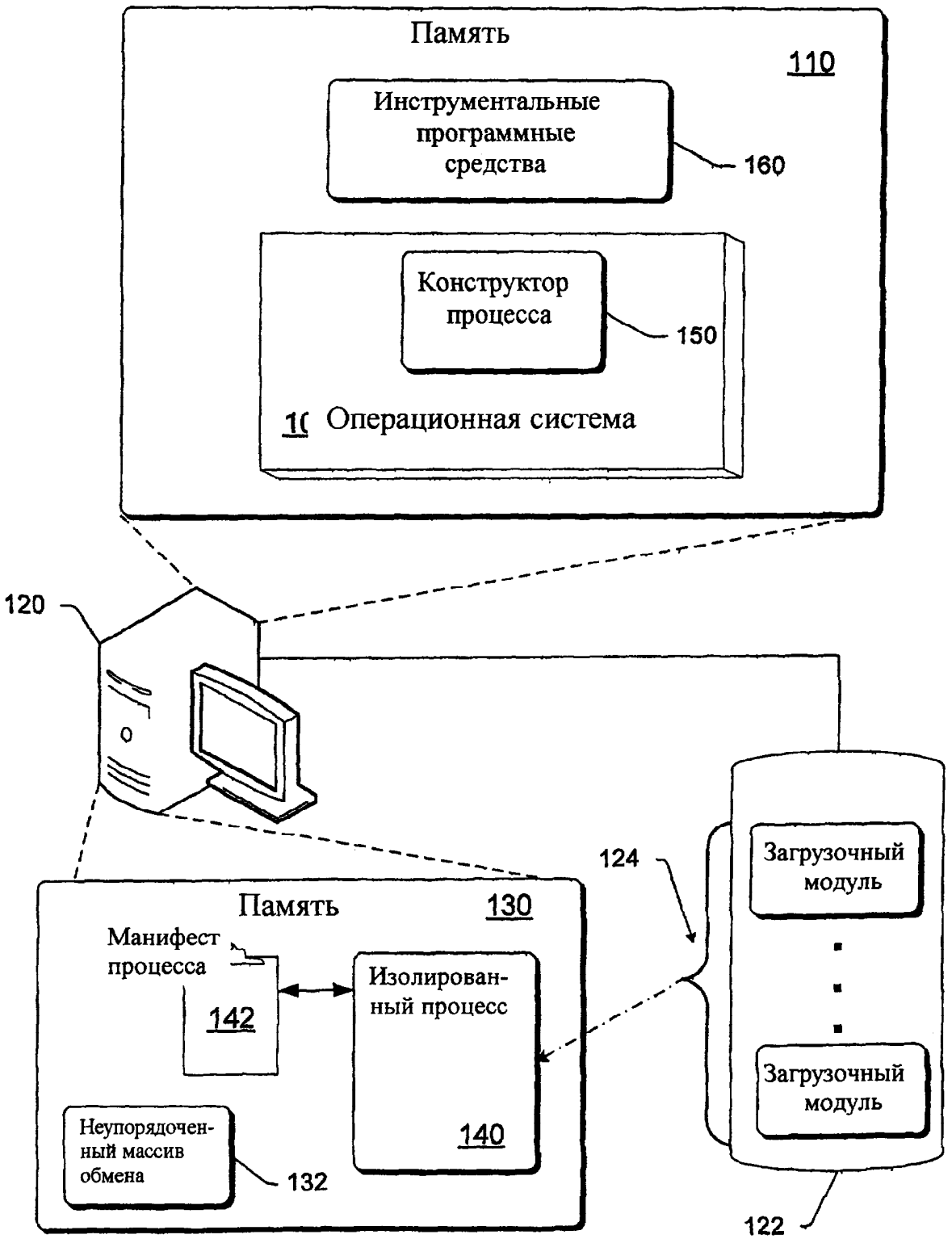
Изобретение относится к области межпроцессного обмена информацией между изолированными процессами. Техническим результатом является повышение эффективности межпроцессного обмена информацией между изолированными процессами. Способ межпроцессного обмена информацией между изолированными процессами состоит в том, что: ассоциативно связывают владение набором данных с первым процессом; отправляют упомянутый набор данных из первого процесса во второй процесс через канал межпроцессного обмена информацией, используя подход без копирования посылаемых данных, причем упомянутый набор данных посылают в соответствии с контрактом канала, ассоциированным с каналом межпроцессного обмена информацией, причем контракт канала статически проверяют во время компиляции до посылки упомянутого набора данных;



RU 2429526 C2

RU 2429526 C2

передают владение упомянутым набором данных от первого процесса второму процессу, при этом первому процессу ограничен доступ к упомянутому набору данных после передачи. 3 н. и 14 з.п. ф-лы, 5 ил.




Фиг. 1

RU 2429526 C2

RU 2429526 C2

RUSSIAN FEDERATION

(19) **RU** (11) **2 429 526** (13) **C2**

(51) Int. Cl.
*G06F 12/00* (2006.01)

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY,
PATENTS AND TRADEMARKS

## (12) ABSTRACT OF INVENTION

(21)(22) Application: **2008116715/08, 16.10.2006**

(24) Effective date for property rights:
**16.10.2006**

Priority:
(30) Priority:
**26.10.2005 US 60/730,546**
**30.06.2006 US 11/428,162**

(43) Application published: **27.10.2009** Bull. 30

(45) Date of publication: **20.09.2011 Bull. 26**

(85) Commencement of national phase: **25.04.2008**

(86) PCT application:
**US 2006/040527 (16.10.2006)**

(87) PCT publication:
**WO 2007/050363 (03.05.2007)**

Mail address:
**129090, Moskva, ul.B.Spasskaja, 25, str.3, OOO "Juridicheskaja firma Gorodisskij i Partnery", pat.pov. A.V.Mitsu, reg.№ 364**

(72) Inventor(s):
**KhANT Galen K. (US),**
**LARUS Dzhejms R. (US),**
**ABADI Martin (US),**
**AJKEN Mark (US),**
**BARKhEhM Pol (US),**
**FANDRICh Manuehl A. (US),**
**KhOBLITTsEL Kris (US),**
**KhODSON Orion (US),**
**LEVI Stiven (US),**
**MERFI Nik (US),**
**STENSGOR B'jarne (US),**
**TARDITI Dehvid (US),**
**UOBBER Tehd (US),**
**ZILL Brajan (US)**

(73) Proprietor(s):
**MAJKROSOFT KORPOREJShN (US)**

RU 2429526 C2

## (54) STATISTICALLY VERIFIED ISOLATED PROCESSES PERMITTING INTER-PROCESS EXCHANGE

(57) Abstract:

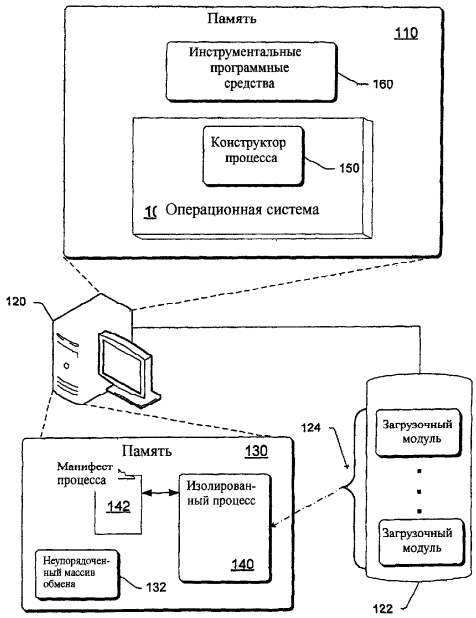
FIELD: information technology.

SUBSTANCE: method for inter-process communication with isolated processes involves: associating ownership of a set of data with a first process, sending said set of data from the first process to a second process through an inter-process communication channel, using an approach without copying the sent data, wherein said set of data is sent in accordance with a channel contract associated with the inter-process communication channel, the channel contract being statistically verified during compilation before sending said set of data; ownership of said set of data is sent from the first process to the second process, where the first process has limited access to said set of data after transfer.

EFFECT: high efficiency of inter-process communication between isolated processes.

17 cl, 5 dwg

RU 2429526 C2




Фиг. *1*

RU 2429526 C2

RU 2429526 C2

## УРОВЕНЬ ТЕХНИКИ

Некоторые операционные системы (ОС) предусматривают изоляцию процессов и межпроцессный обмен информацией. ОС стремятся изолировать процесс так, чтобы он не мог осуществлять доступ к или разрушать данные или выполнять команды другого процесса. В дополнение, изоляция предусматривает четкие границы для завершения процесса и освобождение его ресурсов без взаимодействия с другими процессами. Межпроцессный обмен информацией предоставляет процессам возможность обмениваться данными и сигнализировать о событиях.

Однако есть естественное противоречие между изоляцией и обменом информацией между процессами. Типично, чем более изолированы процессы друг от друга, тем может быть процессам сложнее и потенциально дороже обмениваться информацией друг с другом. Наоборот, чем менее изолированы процессы друг от друга, тем легче процессам обмениваться информацией друг с другом.

Например, процессы, которые совместно используют память, могут считаться имеющими низкую степень изоляции. Процессы с совместно используемой памятью, типично, могут обмениваться информацией, очевидно, более простым способом, только осуществляя запись и чтение непосредственно в/из совместно используемую память. С другой стороны, если ОС не позволяет процессам совместно использовать память, ОС типично предусматривает некоторый механизм, чтобы процессы обменивались информацией.

Из почтения к соображениям производительности компромисс между изоляцией и обменом информацией традиционно решается некоторым образом, который жертвует преимуществами изоляции. В особенности традиционные ОС часто допускают совместное использование памяти между процессами. Так, ОС даже совместно размещают компоненты в пределах того же процесса, максимизируя обмен информацией. Пример такого совместного размещения - это драйверы устройств, расширения браузеров и подключаемые модули веб-служб. Избегание процессной изоляции для такого легкого доступа к таким компонентам может затруднить или уничтожить многие из преимуществ политики изоляции, такие как изоляция сбоев и прозрачное управление ресурсами. Когда одна компонента выходит из строя, сбой часто оставляет совместно используемую память в противоречивом или разрушенном состоянии, что может приводить остальные компоненты в нерабочее состояние.

С другой стороны, по-настоящему изолированные процессы, конечно, используют преимущества политики изоляции. Однако такие изолированные процессы традиционно конфликтуют с межпроцессным обменом информацией.

## СУЩНОСТЬ ИЗОБРЕТЕНИЯ

В материалах настоящей заявки описаны одна или более реализаций операционной системы, которая предусматривает статически проверяемый межпроцессный обмен информацией между изолированными процессами. К тому же в материалах настоящей заявки описана одна или более реализаций инструментальных программных средств, которые облегчают разработку статически проверяемых изолированных процессов, имеющих в распоряжении межпроцессный обмен информацией.

Эта сущность изобретения приведена для ознакомления с вариантами выбора концепций в упрощенном виде, которые дополнительно описаны ниже в подробном описании. Эта сущность изобретения, однако, не подразумевается идентифицирующей ключевые или существенные признаки заявленного объекта изобретения, а также не подразумевается используемой в качестве вспомогательного средства в определении объема заявленного объекта изобретения.

## КРАТКОЕ ОПИСАНИЕ ЧЕРТЕЖЕЙ

По всем чертежам одинаковые номера используются для указания на подобные элементы и признаки.

Фиг. 1 - рабочий сценарий для архитектуры операционной системы, которая поддерживает одну или более реализаций, описанных в материалах настоящей заявки.

Фиг. 2 - другой рабочий сценарий для архитектуры операционной системы, которая поддерживает одну или более реализаций, описанных в материалах настоящей заявки.

Фиг. 3 - структурная схема архитектуры операционной системы, которая поддерживает одну или более реализаций, описанных в материалах настоящей заявки.

Фиг. 4 - блок-схема последовательности операций способа, другой методологической реализации, описанной в материалах настоящей заявки.

Фиг. 5 - блок-схема последовательности операций способа, другой методологической реализации, описанной в материалах настоящей заявки.

## ПОДРОБНОЕ ОПИСАНИЕ

Последующее описание определяет операционную систему (ОС), которая предусматривает изолированные процессы, обладающие возможностью для межпроцессного обмена информацией. Изоляция между изолированными процессами описываемой ОС статически проверяема. Выполняемые команды изолированных процессов могут проверяться во время компиляции, или во время выполнения, или в обоих случаях. К тому же в материалах настоящей заявки описаны одно или более инструментальных средств языков программирования, которые облегчают разработку статически проверяемого межпроцессного обмена информацией между изолированными процессами.

Статически проверяемый процесс является программным процессом, чьи выполняемые команды могут быть проанализированы без действительного исполнения команд процесса. Анализ гарантирует, что процесс не будет себя вести неразрешенными способами и/или мешая работе других процессов или самой операционной системе.

Одна или более реализаций, описанных в материалах настоящей заявки, используют инструменты языка программирования для создания среды, в которой программное обеспечение больше подходит для того, чтобы быть построено лучше, поведение программы легче проверить, и сбои во время выполнения могут быть ограничены и облегчены. Некоторые из признаков одной или более реализаций, описанных в материалах настоящей заявки, в том числе (но не в качестве ограничения):

- данные обмениваются через двунаправленные каналы, где каждый канал состоит точно из двух конечных точек. В любой момент времени каждой конечной точкой канала владеет один поток (т.е. приписывается к одному процессу);
- буферы и другие структуры данных памяти предпочтительно передаются через указатель, а не через копирование данных, которые содержатся в буфере и структурах данных памяти. Эти передачи изменяют владельца блоков памяти;
- канал обмена управляется контрактами статически проверяемого канала, которые описывают сообщения, типы аргументов сообщения и допустимые порядки обмена сообщениями, как конечный автомат с ограниченным числом состояний, подобно сессионным типам;
- конечные точки канала могут быть отправлены в сообщениях через каналы. Таким образом, сеть обмена информации может динамически развиваться;
- отправка и получение в канале не требует выделения памяти;
- отправления являются неблокируемыми и безотказными. Неблокируемость

означает, что отправляющий не ждет, что процесс передачи пройдет успешно. Безотказность означает, что, в конце концов, процесс передачи всегда пройдет успешно. Реализация достигает этого по определению: операция отправки завершается без ожидания результатов. (Однако «канал» может отказать, и это может наблюдаться при приеме по каналу.)

Примерная Операционная Система и Инструментальные Программные Средства

Фиг.1 показывает примерный рабочий сценарий, который поддерживает статически проверяемые межпроцессно обменивающиеся информацией Программно-Изолированные Процессы (SIP) и использование инструментальных программных средств, которые облегчают программирование таких статически проверяемых межпроцессно обменивающихся информацией SIP.

Фиг.1 показывает операционную систему 100 и инструментальные программные средства 160, сохраненные и/или выполняемые в памяти 110 компьютера 120. Компьютер 120, типично, содержит множество удобочитаемых процессором носителей (включая память 110). Такие носители могут быть любыми имеющимися в распоряжении компьютера 120 носителями, к которым может быть осуществлен доступ компьютером, и включают в себя как энергозависимые и энергонезависимые носители, так и съемные и несъемные носители.

Компьютер 120 включает в себя компьютерное запоминающее устройство 122 (например, жесткий диск, RAID систему и т.д. ), которое сохраняет набор загруженных модулей 124, и рабочую память 130 (которая может быть частью или быть отдельно от памяти 110).

Рабочая память 130 также включает неупорядоченный массив обмена 132, который является буфером, используемым для хранения информации (такой, как указатели на расположение в рабочей памяти 130). В материалах настоящей заявки неупорядоченный массив обмена может называться "буфером", "совместно используемым буфером обмена" или как-то эквивалентно этому. Неупорядоченный массив включает составные адресуемые блоки памяти (как показано на блоках 134). Несмотря на то что неупорядоченный массив обмена 132 как целое является доступным для различных процессов, каждым индивидуальным блоком владеет один процесс в один момент времени (когда этот блок используется). Однако владение блоком памяти может быть изменено на владение другим активным процессом. Таким образом, этим способом неупорядоченный массив обмена 132 предусматривает механизм обмена данными для SIP.

Как изображено, операционная система 100 содержит в себе модуль конструктора 150 процессов. Конструктор процессов может быть частью ядра операционной системы 100. Конструктор процессов 150 создает процессы в рабочей памяти компьютера из динамического набора образующих компонент, которые типично объявляются как набор загрузочных модулей в запоминающем устройстве компьютера.

В примере на Фиг.1 конструктор 150 процессов создает процесс 140, который сохраняется в рабочей памяти 130. Как изображено здесь, процесс 140 создается из загрузочных модулей 124, которые являются реализующими образующими компонентами процесса, объединяемых этими расширяющими процесс компонентами.

Процесс 140 обладает манифестом процесса 142, который определяет содержание процесса 140, разрешаемое поведение процесса и другие возможные свойства процесса. Как изображено здесь, манифест 142 процесса напрямую связан с процессом (таким, как процесс 140), чью структуру он описывает.

Инструментальные программные средства 160 содержат в себе модули и структуры данных. В связи с этим инструментальные программные средства 160 содействуют лицам, которые разрабатывают процесс, в создании статических переменных и изолированного процесса с оговоренным и ограниченным межпроцессным обменом информацией процесса. Инструментальные программные средства 160 облегчают эту разработку посредством использования наложения сильных инвариантов, которые принудительно применяются в момент компиляции, выполнения или в обоих случаях. Сильные инварианты описаны ниже в разделе «Проверка».

Инструментальные программные средства 160 предусматривают средства статического анализа, которые помогают программистам находить, исправлять и/или предупреждать ошибки межпроцессного обмена информацией без затратного по времени тестирования и отладки. Посредством увеличения эффективности и применимости средств детерминистического статического предвычислительного анализа инструментальные программные средства 160 также увеличивают вероятность того, что программист или группа программистов будет выпускать программу или набор программ, свободных от ошибок, которые связаны с межпроцессным обменом информацией, и также сокращают тестирование и отладку, которые требуют усилий при выпуске такой программы или набора программ.

Описанные инструментальные программные средства (например, инструментальные программные средства 160 на Фиг. 1) используют программные конструкторы и подходы, которые облегчают разработчикам использование и создание SIP (как описано в материалах настоящей заявки). С описанными инструментальными программными средствами обмен информации SIP может быть статически проверен.

Программно Изолированный Процесс

В области компьютерной науки и наиболее часто в технологии операционных систем термин «программный процесс» (или более просто «процесс») хорошо известен. Приложения часто состоят из одного или более процессов. Операционная система (ОС) знает и, несомненно, может управлять и контролировать один или более отдельных процессов, выполняемых на компьютере.

Одна или более реализаций, описываемых в материалах настоящей заявки, функционируют в модели ОС, которая предусматривает и/или поддержку абстрактной модели Программно Изолированного Процесса (SIP). SIP инкапсулируют части программы или системы и обеспечивают сокрытие информации, изоляцию сбоев, и строгие интерфейсы. SIP используются везде в ОС и в прикладном программном обеспечении в соответствии с описанными реализациями.

Что касается SIP, то выполняемый код вне ядра выполняется в SIP и взаимодействует через строго типизированный канал взаимодействия. SIP - это закрытое окружение, которое не допускает совместное использование данных или динамическую загрузку кода. SIP отличаются от традиционных процессов ОС в нескольких аспектах. Далее представлены примеры таких аспектов SIP, отличных от традиционных процессов ОС:

- SIP - это закрытые объектные пространства, а не адресные пространства. Два SIP не могут одновременно получить доступ к объекту. Обмены информацией между процессами изменяют исключительное владение данными.

- SIP - закрытые пространства кода. Процесс не может динамически загружать или генерировать код.

- SIP не используют для изоляции аппаратное управление памятью, так,

несколько SIP могут находиться в физическом или виртуальном адресном пространстве.

- Обмены информации между SIP осуществляются через двунаправленные, строго типизированные, высокоуровневые каналы. Тип канала описывает его протокол обмена информации, а также значения, которые он передает, и эти оба аспекта проверяются.

- SIP недорого создавать, и расходы на межпроцессный обмен информацией между SIP малозначительны. Их низкая стоимость делает практичным использование SIP как более детальный уровень изоляции и механизм расширения.

- SIP создаются и управляются операционной системой, так что при завершении ресурсы SIP могут быть эффективно возвращены.

- SIP не зависят от среды выполнения, даже от расширений имеющих различные макеты данных, систем выполнения и «сборщиков мусора». Другие безопасные языковые системы поддерживают одну среду выполнения.

Термин «Программно изолированные процессы» или SIP используются здесь для удобства. Это не сужает рамки этой концепции. Несомненно, эта концепция может быть реализована в программных, аппаратных средствах, микропрограммном обеспечении или их сочетании.

Межпроцессный обмен информацией

Фиг. 2 иллюстрирует примерную архитектуру 200 межпроцессного обмена информацией (IPC), которая облегчает межпроцессный обмен информацией без непредвиденных взаимодействий между SIP. В дополнение к предусматриванию обмена информацией между процессами примерная IPC архитектура 200 может предусматривать обмен информацией между процессами и ядром операционной системы.

Используя примерную IPC архитектуру 200, SIP обмениваются информацией, исключительно отправляя сообщения через каналы, которые являются двунаправленными, типизированными соединениями между двумя процессами. К сообщениям приложены наборы значений или блоки сообщения в «Неупорядоченном массиве Обмена» (как, например, неупорядоченный массив обмена 132 выше на Фиг. 1), которые передаются из отправляющего процесса к принимающему. Канал типизирован контрактом, который точно устанавливает формат сообщений и допустимые последовательности сообщений в канале.

Как изображено на Фиг. 2, примерная IPC архитектура 200 реализована на компьютере 202, который сконфигурирован памятью 210 (например, энергозависимой, энергонезависимой, съемной, несъемной и т.д.). Операционная система (ОС) 212 показана как сохраненная в памяти 210 и выполняемая на компьютере 202.

ОС 212 имеет ядро 220. Ядро 220 ОС содержит в себе координатора 222 Межпроцессного Обмена Информацией (IPC). Ядро 220 ОС может создавать один или более процессов. Фиг. 2 показывает, например, три активных процесса (230, 240, и 250), запущенных в памяти 210.

IPC координатор 222 способствует обменам информации между активными процессами (например, процессы 230, 240 и 250). В то время как Фиг. 2 иллюстрирует ядро 220 ОС, реализующей IPC координатор 222, другие реализации могут иметь IPC координатора, который находится вне ядра ОС. В таком случае каждый посредник непременно работает во взаимодействии и координации с ОС.

Память 210 также включает в себя неупорядоченный массив обмена 290, который

имеет несколько блоков памяти 292. Неупорядоченный массив обмена 290 доступен нескольким активным процессам (например, процессам 230, 240 и 250). Это обеспечивает для SIP механизм обмена данными.

«Межпроцессные обмены информацией, Использующие Двунаправленные средства Передачи Сообщений» раскрывают дополнительные детали относительно примерной IPC архитектуры 200, которая применима для одной или нескольких реализаций, описанных в материалах настоящей заявки.

Неупорядоченный массив обмена

Каждый SIP поддерживает свои собственные независимые и персональные неупорядоченные массивы. SIP совместно не используют память друг с другом. Так, когда данные помещаются из одного SIP в другой SIP, передаваемые данные не приходят из приватного неупорядоченного массива процесса. Вместо этого данные приходят из обособленного неупорядоченного массива, который используется для сохранения данных, которые передаются между процессами. Этот отдельный неупорядоченный массив - это неупорядоченный массив обмена, например неупорядоченный массив обмена 132, показанный на Фиг. 1, или неупорядоченный массив обмена 290, показанный на Фиг. 2.

SIP могут содержать указатели на их собственный персональный неупорядоченный массив. В дополнение, SIP могут содержать указатели на публичный неупорядоченный массив обмена. По меньшей мере, в одной описанной реализации неупорядоченный массив обмена содержит только указатели на самого себя. Каждый из SIP может хранить множество указателей на неупорядоченный массив обмена. Однако каждый блок памяти в неупорядоченном массиве обмена принадлежит (то есть доступен) не более чем одному SIP в любой момент времени во время выполнения системы.

Когда выполняется статическая проверка (верификация), инструментальные программные средства 160 могут отслеживать владельца блоков памяти в неупорядоченном массиве обмена, потому что каждым блоком владеет не более чем один процесс в любой момент времени. Факт того, что каждый блок в неупорядоченном массиве обмена доступен одному процессу в любой момент времени, также предусматривает полезную гарантию взаимного исключения.

Каналы

С IPC архитектурой 200 канал является двунаправленным средством передачи, состоящим в точности из двух конечных точек. Конечные точки иногда называются равноправными участниками канала. Канал доставляет сообщения с минимальными потерями и по порядку. К тому же сообщения типично извлекаются по порядку, как они были отправлены. Семантически каждая конечная точка имеет очередь приема, и отправка в конечной точке ставит в очередь сообщение для равноправного участника.

Каналы описываются контрактами каналов. Другими словами, контракт каждого канала определяет ограничения межпроцессных обменов информацией по этому каналу. Например, контракт может устанавливать, с какими другими процессами этот процесс может общаться и как такое общение может происходить. Два конца канала являются, типично, несимметричными. Для целей этого описания одна конечная точка называется импортирующим концом (Imp) и другая - экспортирующим концом (Exp). Они отличаются по уровню типов, что касается типов C_Imp и C_Exp соответственно, где C - контракт канала, управляющий взаимодействием.

Фиг. 2 образно иллюстрирует каналы как электрические вилки, провода и розетки. По меньшей мере, в одной описанной реализации каналы имеют точно только две конечные точки и каждая конечная точка принадлежит не более чем одному каналу.

Как изображено, канал 260 соединяет процесс 230 и ядро 220 ОС и обладает только двумя конечными точками 262 и 264. Канал 270 соединяет процесс 240 и процесс 250 и обладает только двумя конечными точками 272 и 274. Канал 280 является вновь созданным каналом, который первоначально соединяет процесс 250 с собой, но, по-прежнему, обладает только двумя конечными точками 282 и 284.

Эти каналы представлены в графическом образе «электрических проводов» с точно двумя «вилками» (представляющими конечные точки). Более предпочтительно, чем передача электрического тока, представлять, что эти «провода» передают сообщения, отравляемые и принимаемые каждым участником («двунаправленность»), где «провод» вставлен в розетку. Это прохождение двунапрвленного сообщения проиллюстрировано направленными конвертами около канала 270.

IPC архитектура 200 предлагает передающий сообщения IPC механизм обмена информации. Вместо использования своевременной записи и чтения некоторой совместно используемой памяти (как в традиционных подходах) IPC архитектура 200 ограничивает межпроцессные обмены информацией отправкой и получением сообщений.

Традиционные передающие сообщения подходы ОС являются только односторонними механизмами - часто с любым одним отправителем и несколькими получателями или несколькими отправителями и одним получателем. В отличие от этих традиционных подходов канал IPC архитектуры 200 является двухсторонним механизмом с точно двумя конечными точками и не более участниками.

Это иллюстрируется каналом 260 и каналом 270 на Фиг. 2. Канал 260 соединяет процесс 230 и ядро 220 ОС и только их двоих. Канал 270 соединяет процесс 240 и процесс 250 и только их двоих.

Как проиллюстрировано на Фиг. 2, каждый из двунаправленных IPC каналов имеет точно две конечные точки канала. Каждой конечной точкой канала владеет не более чем один процесс одновременно. Например, одной конечной точкой канала владеет один процесс, а другая конечная точка принадлежит другому процессу, или ей владеет ядро операционной системы. Конечные точки могут перемещаться через каналы. При этом владение этой конечной точкой передается.

IPC координатор 222 гарантирует, что каждым сообщением и каждой инкапсуляцией сообщения владеет не более чем один процесс в любой момент. Это может выполняться посредством использования абстракции уровня канала для каждого канала. Более того, на канальном уровне абстракции сообщение находится в памяти, доступной не более чем одному процессу в любой момент. С точки зрения процессов, обменивающихся информацией, состояние, содержащееся в или достижимое из сообщения, никогда совместно не используется. По меньшей мере, в одной описанной реализации сообщения доступны создателю сообщения только до тех пор, пока он их не отправил. По меньшей мере, в одной описанной реализации сообщения доступны получателю сообщения только после того, как он их получил.

Владение

Изоляция памяти конечных точек и других данных, передаваемых в канале, гарантируется через отслеживание во время компиляции всех блоков в неупорядоченном массиве обмена. В частности, статические проверки обязывают, чтобы обеспечение доступом к их ресурсам происходило в точках программы, которые ресурсами владеют, и такие методы не дают потери владения ресурсами. Отслеживаемые ресурсы имеют строгую модель владения.

Каждым ресурсом владеет не более чем один процесс в любой момент времени.

Например, если конечная точка пересылается в сообщении из потока T1 в поток T2, то владение конечной точкой изменяется: от T1 к сообщению и затем к T2 после приема сообщения.

В традиционных подходах процесс делает копию данных и передает эти данные. Поэтому этими данными сейчас владеют несколько процессов. Процесс, который посылает данные, может по-прежнему влиять на свою копию данных.

По меньшей мере, в одной описанной реализации владение данными связывается со специфичными SIP. Владение данными передается, когда передаются данные. Поэтому отправляющий SIP не может подействовать на данные непосредственно сразу после их передачи, SIP больше не имеет доступа к ним и не может сделать их копию. В одной или нескольких реализациях, описанных в материалах настоящей заявки, данными владеет один SIP и их владение передается вместе с данными сразу, как только они отправляются через канал.

Подобным образом, каждой конечной точкой канала владеет только один SIP. Владение конечной точкой передается вместе передачей конечной точки другому SIP. Как только оно отправляется, отправляющий SIP больше не имеет доступа к конечной точке канала, которую он только что послал.

Эта передача владения (конечной точкой и данными) достигается посредством неупорядоченного массива обмена, например неупорядоченного массива обмена 132, показанного на Фиг. 1, или неупорядоченного массива обмена 290, показанного на Фиг. 2. Более точно, блок памяти в неупорядоченном массиве обмена содержит указатель (на другое место в памяти рассматриваемых данных или рассматриваемой конечной точки). При обмене с другим процессом через канал отправляющий процесс передает указатель на блок памяти в неупорядоченном массиве обмена принимающему процессу.

Этим способом отправляющий процесс эффективно передает рассматриваемые данные принимающему процессу, но делает это без создания или сохранения копии для себя. Более того, отправляющий процесс эффективно передает владение рассматриваемой конечной точки к принимающему процессу без сохранения владения. Переход владения может также быть описан таким образом, как передача отправителем сообщения владения, сохраняя указатель на сообщение в конечной точке получателя, в оговоренном месте через текущее состояние протокола обмена сообщениями.

Эти обмены, где нет копируемых данных, могут быть названы подходом «нулевого копирования». Используя такой подход, буферы диска и сетевые пакеты могут передавать через различные каналы, через стек протоколов и в процессы приложения без копирования или любого сохранения посылаемых данных.

Контракты каналов

Контракты каналов используются реализациями, описанными в материалах настоящей заявки, для того, чтобы облегчить архитектуру изоляции процессов. Контракты каналов (и другие аспекты межпроцессного обмена информацией) также описаны в «Межпроцессные обмены информацией, Использующие Двунаправленные средства Передачи Сообщений».

Здесь представлен пример контракта, описывающий простое взаимодействие в канале.

```
contract C1 {
  in message Request(int x) requires x>0;
  out message Reply(int y);
  out message Error();

  state Start: Request?
               -> (Reply! or Error!)
             * -> Start;
}
```

В этом примере контракт C1 объявляет три сообщения: Request, Reply и Error. Каждое объявление сообщений задает тип аргументов, содержащихся в сообщении. Например, оба Request и Replay содержат одно целое значение, тогда как Error не передает каких либо значений. Дополнительно, каждое сообщение может точно задавать оператор Spec#, дополнительно ограничивающий аргументы.

Сообщения могут так же быть помечены направлением. Контракт записывается с точки зрения экспортера. Таким образом, в этом примере Request - это сообщение, которое может быть отправлено импортером к экспортеру, тогда как Reply и Error оправляются от экспортера к импортеру. Без описателя сообщения могут отправляться в обоих направлениях.

После объявления сообщения контракт устанавливает допускаемые взаимодействия сообщений в виде конечного автомата, управляемого событиями отправки и получения. Первое объявленное состояние считается исходным состоянием сообщения. В примере контракта C1 объявляет единственное состояние, называемое Start. После имени состояния действие Request показывает, что в состоянии Start экспортная сторона канала готова получить сообщение Request. Идущая следом конструкция (Replay! or Error!) устанавливает, что экспортер пошлет (!) либо сообщение Replay, либо Error. Последняя часть (-> Start) устанавливает, что взаимодействие затем возобновится с состояния Start, таким образом, цикл продолжится до бесконечности.

Незначительно более сложный пример - это часть контракта для сетевого стека.

```
public contract TcpConnectionContract {
  // Requests
  in message Connect(uint dstIP,
     ushort dstPort);

  out message Ready();

  // Initial state
  state Start : Ready! -> ReadyState;

  state ReadyState : one {
      Connect? -> ConnectResult;
      BindLocalEndPoint? -> BindResult;
      Close? -> Closed;
  }

  // Binding to a local endpoint
  state BindResult : one {
      OK! -> Bound;
      InvalidEndPoint! -> ReadyState;
  }

  in message Listen();

  state Bound : one {
      Listen? -> ListenResult;
      Connect? -> ConnectResult;
      Close? -> Closed;
  }
...
```

Спецификация протокола в контракте служит нескольким целям. Она может помочь обнаружить программные ошибки как в момент исполнения, так и с помощью средств статического анализа. Мониторинг во время исполнения управляет конечным автоматом контракта в ответ на обмен сообщениями по каналу и следит за ошибочными переходами. Само собой, техника мониторинга во время исполнения замечает ошибки в выполнении одной программы, но не может заметить ошибки «живучести», например отсутствие завершения. Свойство живучести - это свойство в формулировке «что-то хорошее случится в конце концов», например «в конце концов программа отправит сообщение». Статический программный анализ может предоставить большую гарантию того, что процесс корректный и свободен от ошибок зависания в исполняемой программе. В общем случае статический анализ не ограничивается контролем одной исполняемой программы, как это происходит. Возможно, например, полагаться на проверку команд процесса для того, чтобы определить - будет или нет процесс что-то делать в итоге. Существуют основные результаты в логике, которые говорят, это не всегда будет работать, но это может

работать достаточно хорошо во многих случаях.

Одна реализация использует комбинацию контроля во время исполнения и статическую проверку. Все сообщения в канале проверяются контрактом канала, который обнаруживает правильность, но не проблемы живучести. Реализация, описанная в материалах настоящей заявки, имеет статическую программу контроля, которая проверяет свойства безопасности.

В дополнение, компилятор использует контракт, чтобы определять максимальное число сообщений, которые могут находиться в канале, что позволяет компилятору статически распределять буферы в конечных точках канала. Статическое распределение буферов улучшает производительность обмена информации.

Конечные точки

Каналы представлены как пары конечных точек, представляющих импортирующую и экспортирующую сторону канала. У каждой точки есть тип, который устанавливает контракт канала. Типы конечных точек неявно объявляются в каждом контракте. Контракт C1 представляет собой класс, а типы конечных точек являются вложенными типами внутри этого класса, как изложено ниже:

- C1.Imp - тип импортирующей конечной точки канала с контрактом C1.
- C1.Exp - тип экспортирующей конечной точки канала с контрактом C1.

Методы Send/Receive

Каждый класс контракта содержит методы отправки и получения сообщений, объявленных в контракте. В пример предусматриваются следующие методы:

```
C1.Imp {
  void SendRequest(int x);
  void RecvReply(out int y) ;
  void RecvError();
}

C1.Exp {
  void RecvRequest(out int x)
  void SendReply(int y);
  void SendError();
}
```

Семантика методов Send заключаются в том, что они посылают сообщения асинхронно. Методы приема блокируются до тех пор, пока данное сообщение не прибудет. Если другое сообщение прибудет первым, то возникнет ошибка. Такие ошибки не должны никогда появляться, если программа проходит тест на проверку контракта. Пока получатель точно не знает, какое сообщение требуется следующим, эти методы не подходят.

Методологические реализации

Фиг. 3 показывает способы 300 и 400 для облегчения эффективного межпроцессного обмена информацией статически проверяемых SIP. Эти способы 300 и 400 выполняются одним или более различными компонентами, как изображено на Фиг. 1 и 2. Более того, эти способы 300 и 400 могут быть исполнены на программных, аппаратных средствах, программно-аппаратном обеспечении или их сочетании.

На этапе 302 на Фиг. 3 операционная система (ОС) предусматривает выполнение

одного или более программно изолированных процессов (SIP) в среде операционной системы компьютера.

На этапе 304 ОС ассоциативно связывает владение конкретным набором данных с первым SIP. Этот набор данных может быть блоком памяти в неупорядоченном массиве обмена, например в неупорядоченном массиве обмена 132, показанном на Фиг. 1, или неупорядоченном массиве обмена 290, показанном на Фиг. 2. Этот набор данных может быть сообщением. Этот набор данных может включать в себя данные или один или более указателей на место в памяти, содержащее данные. Также этот набор данных может включать один или более указателей на конечную точку канала.

На этапе 306 ОС посылает конкретный набор данных от первого SIP ко второму SIP. Посылка может состоять из предоставления указателей на набор данных (в неупорядоченном массиве обмена) второму SIP. В качестве альтернативы посылка может состоять из записи сообщений в конечную точку канала, соединенного со вторым SIP.

На этапе 308 ОС передает владение конкретным набором данных от первого SIP ко второму SIP. Когда сообщение отправлено по каналу, владение переходит от отправляющего SIP к принимающему SIP. Отправляющий SIP больше не сохраняет ссылку на сообщение. В действительности, отправляющий SIP больше не имеет доступа к отправленному сообщению.

В процессе отправки 306 и передачи 308 никакой копии отправленной информации не сохраняется. В самом деле, никакой копии отправленной информации не создается. Непосредственно после того, как указатель на набор данных (или более точно, указатель на блок памяти, сохраняющей данные или указатель) передается, никакая копия не создается и не пересылается.

Этот инвариант владения навязывается инструментальными программными средствами и операционной системой (например, инструментальными программными средствами 160 и ОС 100). Этот инвариант владения служит, по меньшей мере, трем целям: первое - это предотвращение совместного использования между процессами. Второе - это облегчение статического программного анализа за счет исключения совпадения имен сообщений. Третье - это разрешение реализации гибкости посредством предусматривания передающих сообщения семантик, которые могут быть реализованы копированием или передачей указателей.

Как изображено на Фиг. 4, на этапе 402 операционная система (ОС) предусматривает выполнение одного или более программно изолированных процессов (SIP) в операционной системе компьютера.

На этапе 404 ОС ассоциативно связывает владение конкретной конечной точкой конкретного канала межпроцессных обменов информацией с первым SIP. Этот набор данных может быть блоком памяти в неупорядоченном массиве обмена, например в неупорядоченном массиве обмена 132, показанном на Фиг. 1, или неупорядоченном массиве обмена 290, показанном на Фиг. 2. Этот набор данных может быть сообщением. Этот набор данных может включать в себя один или более указателей. Этот набор данных может включать в себя один или более указателей на место в памяти, содержащее один или более указателей. Также этот набор данных может включать в себя один или более указателей на конечную точку канала.

На этапе 406 ОС посылает конкретную конечную точку конкретного канала межпроцессных обменов информацией из первого SIP во второй SIP. Посылка может состоять из предоставления указателей на конкретные указатели (в неупорядоченном массиве обмена) второму SIP. В качестве альтернативы посылка может состоять из

записи сообщений в конечную точку канала, соединенного со вторым SIP.

На этапе 408 ОС передает владение конкретной конечной точкой конкретного канала межпроцессных обменов информацией из первого SIP во второй SIP. Когда владение конечной точкой передается из отправляющего SIP к принимающему SIP, отправляющий SIP больше не сохраняет ссылку на сообщение. В действительности, отправляющий SIP больше не имеет доступа к отправленным данным.

Более того, эта передача владения конечной точкой происходит без создания и передачи «копии». Непосредственно после того, как указатель на конечную точку (или, более точно, указатель на блок памяти, сохраняющей конечную точку) передается, никакая копия не создается и не пересылается.

Проверка

Инструментальные программные средства 160 могут проверять (верифицировать) программирование одного или более SIP. Инструментальные программные средства 160 контролируют, что выполняемый код имеет типовую безопасность и принудительно использует строгие инварианты компилятором и во время выполнения. Такие строгие инварианты включают в себя (в виде примера, а не ограничения):

- каждый блок в неупорядоченном массиве обмена имеет не более чем один собственный поток (то есть процесс) в любой момент времени;

- блоки в неупорядоченном массиве обмена доступны только собственнику блока. Таким образом, отсутствует доступ после того, как блок будет освобожден или передано владение;

- соблюдение контракта канала, который определяет и ограничивает межпроцессный обмен информацией между процессами (например, последовательность сообщений, наблюдаемое в канале, соответствует контракту канала).

Методологическая реализация проверки

Фиг. 5 показывает способ 500 для проверки изолированных процессов. Эти способы 500 и 400 выполняются одним или более различными компонентами, как изображено на Фиг. 1 и 2. Более того, эти способы 500 и 400 могут быть исполнены на программных, аппаратных средствах, программно-аппаратном обеспечении или их сочетании.

На этапе 502 на Фиг.5 компилируется выполняемый код для одного или более программно-изолированных процессов (SIP) на окружении операционной системы компьютера, которая поддерживает SIP.

На этапе 504 в процессе компиляции инструментальные программные средства подтверждают, что каждый блок памяти в неупорядоченном массиве обмена имеет не более чем один владеющий процесс в любой момент времени. Это означает, что только один SIP будет владеть любым конкретным блоком памяти в любой момент времени.

На этапе 506 в процессе компиляции инструментальные программные средства подтверждают, что каждый блок памяти в неупорядоченном массиве обмена доступен только законному владельцу (например, SIP).

На этапе 508 в процессе компиляции инструментальные программные средства 160 подтверждают, что контракты канала выполнены. Например, инструментальные средства подтверждают, что последовательность сообщений, определенная для контроля, соблюдается.

Инструментальные программные средства 160 могут предоставлять отчет о

результатах такого подтверждения пользователям, программным модулям и/или операционной системе. Инструментальные программные средства 160 могут выполнять эту проверку в процессе компиляции. В дополнение, они могут также проверять те же самые свойства на сгенерированном промежуточном языке. Более того, инструментальные программные средства 160 еще раз могут проверять результирующую форму типизированного ассемблера.

**Вывод**

Технологии, описанные в материалах настоящей заявки, могут быть реализованы различными способами, в том числе (но не в качестве ограничения) программными модулями, вычислительными системами общего и специального назначения, сетевыми серверами и специализированной электроникой и аппаратными средствами, аппаратно реализованным программным обеспечением, в качестве части одной или более компьютерных сетей или их сочетанием.

Одна или более реализаций, описанных в материалах настоящей заявки, могут быть реализованы с помощью многих широко известных вычислительных систем, сред и/или конфигураций, которые пригодны для использования, в том числе, но не в качестве ограничения, персональных компьютеров (ПК, PC), серверных компьютеров, карманных или портативных устройств, многопроцессорных систем, основанных на микропроцессорах систем, программируемой бытовой электроники, беспроводных телефонов и оборудования, приборов общего применения и специального назначения, специализированных интегральных схем (ASIC), сетевых ПК, «тонких клиентов», «толстых клиентов», телевизионных абонентских приставок, миникомпьютеров, универсальных вычислительных машин, распределенных вычислительных сред, которые включают в себя любую из вышеприведенных систем или устройств, и тому подобное.

Хотя одна или более вышеописанных реализаций были описаны на языке, специфичном для конструктивных признаков и/или методологических этапов, должно быть понятно, что другие реализации могут быть осуществлены на практике без специфичных примерных признаков или этапов, описанных в материалах настоящей заявки. Точнее, специфичные примерные признаки и этапы раскрыты в качестве предпочтительных форм одной или более реализаций. В некоторых случаях хорошо известные свойства могли быть опущены или упрощены для того, чтобы прояснить описание примерной реализации. Более того, для облегчения понимания определенные этапы способов схематически изображены как отдельные этапы; однако эти отдельные схематически изображенные этапы не должны истолковываться в качестве обязательно зависимых от очередности при их выполнении.

## Формула изобретения

1. Считываемый процессором носитель, имеющий выполняемые процессором команды, которые, когда выполняются процессором, исполняют способ межпроцессного обмена информацией между изолированными процессами, состоящий в том, что:

ассоциативно связывают владение набором данных с первым процессом;

отправляют упомянутый набор данных из первого процесса во второй процесс через канал межпроцессного обмена информацией, используя подход без копирования посылаемых данных, причем упомянутый набор данных посылают в соответствии с контрактом канала, ассоциированным с каналом межпроцессного обмена информацией, причем контракт канала статически проверяют во время компиляции до

посылки упомянутого набора данных;

передают владение упомянутым набором данных от первого процесса второму процессу, при этом первому процессу ограничен доступ к упомянутому набору данных после передачи.

2. Носитель по п.1, в котором набор данных включает в себя сообщение.

3. Носитель по п.1, в котором канал межпроцессного обмена информацией ограничен иметь одну исходную конечную точку.

4. Носитель по п.1, в котором передача владения происходит через канал межпроцессного обмена информацией, соединяющий первый процесс и второй процесс, при этом первый процесс и второй процесс находятся в одном и том же виртуальном или физическом адресном пространстве.

5. Носитель по п.1, в котором как отправление, так и передача выполняются без выделения памяти.

6. Носитель по п.1, в котором набор данных сохраняется в адресуемом местоположении в распределенной памяти, причем распределенная память имеет несколько адресуемых местоположений, каждое местоположение доступно либо первому, либо второму процессу, но не обоим одновременно.

7. Считываемый процессором носитель, содержащий выполняемые процессором команды, которые при выполнении процессором исполняет способ межпроцессного обмена информацией между изолированными процессами, состоящий в том, что:

ассоциативно связывают с первым процессом владение конечной точкой канала межпроцессного обмена информацией,

отправляют конечную точку от первого процесса ко второму процессу через канал межпроцессного обмена информацией в соответствии с контрактом статически проверяемого канала, ассоциированным с упомянутым каналом межпроцессного обмена информацией, причем контракт статически проверяемого канала подтверждает во время компиляции, что конечной точкой одновременно владеет только один процесс; передают владение конечной точкой от первого процесса ко второму процессу, используя подход без копирования посылаемых данных, так что первый процесс не сохраняет копию конечной точки после передачи.

8. Носитель по п.7, в котором первый процесс уже не осуществляет доступ к конечной точке после передачи.

9. Носитель по п.7, в котором посылка конечной точки от первого процесса второму процессу включает в себя посылку указателя на адресуемое местоположение в распределенной памяти, причем конечная точка сохранена в упомянутом адресуемом местоположении.

10. Носитель по п.9, в котором адресуемое местоположение доступно только одному процессу одновременно.

11. Носитель по п.7, в котором как отправление, так и передача выполняются без выделения памяти.

12. Носитель по п.7, в котором посылка конечной точки включает в себя посылку сообщения от первого процесса второму процессу через упомянутый канал межпроцессного обмена информацией.

13. Считываемый процессором носитель, содержащий выполняемые процессором команды, которые при выполнении процессором исполняет способ межпроцессного обмена информацией между изолированными процессами, состоящий в том, что:

создают один или более изолированных программных процессов в среде операционной системы компьютера, при этом полученные два или более

изолированных программных процесса отформатированы так, чтобы быть исполняемыми в среде операционной системы компьютера;

посылают посредством посылающего изолированного программного процесса данные по каналу, описываемому контрактом статически проверяемого канала, причем упомянутый канал является двунаправленным каналом, имеющим точно две конечные точки, состоящие из первой конечной точки, которой владеет первый изолированный программный процесс, и второй конечной точки, которой владеет принимающий изолированный программный процесс, при этом каждой из двух конечных точек владеет самое большее один из изолированных программных процессов одновременно, и

передают исключительное владение данными от посылающего изолированного программного процесса к принимающему изолированному программному процессу, как только данные посланы по каналу,

причем посылающий изолированный программный процесс посылает данные на принимающий изолированный программный процесс без создания копии этих данных.

14. Носитель по п.13, в котором посылка данных включает в себя посылку указателя на блок памяти совместно используемого неупорядоченного массива обмена от посылающего изолированного программного процесса к принимающему изолированному программному процессу, причем упомянутый блок памяти доступен для более одного изолированных программных процессов одновременно.

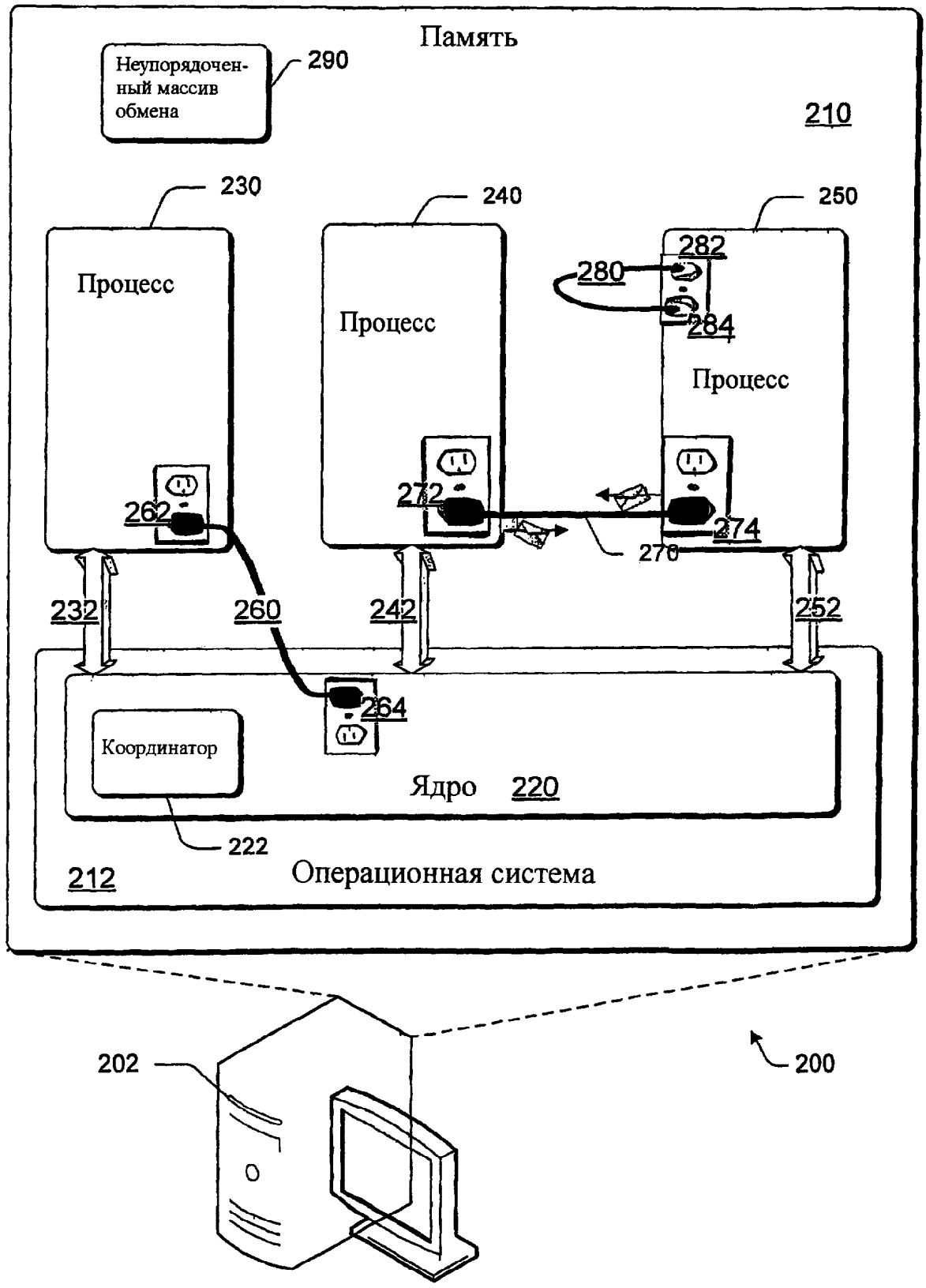
15. Носитель по п.13, в котором изолированные программные процессы являются Изолированными Программными Процессами (SIP), причем способ дополнительно содержит подтверждение во время выполнения, что межпроцессный обмен через упомянутый канал соответствует контракту статически проверяемого канала.

16. Носитель по п.13, в котором изолированные программные процессы являются Изолированными Программными Процессами (SIP), причем Изолированные Программные Процессы являются независимыми средами выполнения, имеющими одно или более из различных макетов данных, систем выполнения и «сборщиков мусора».

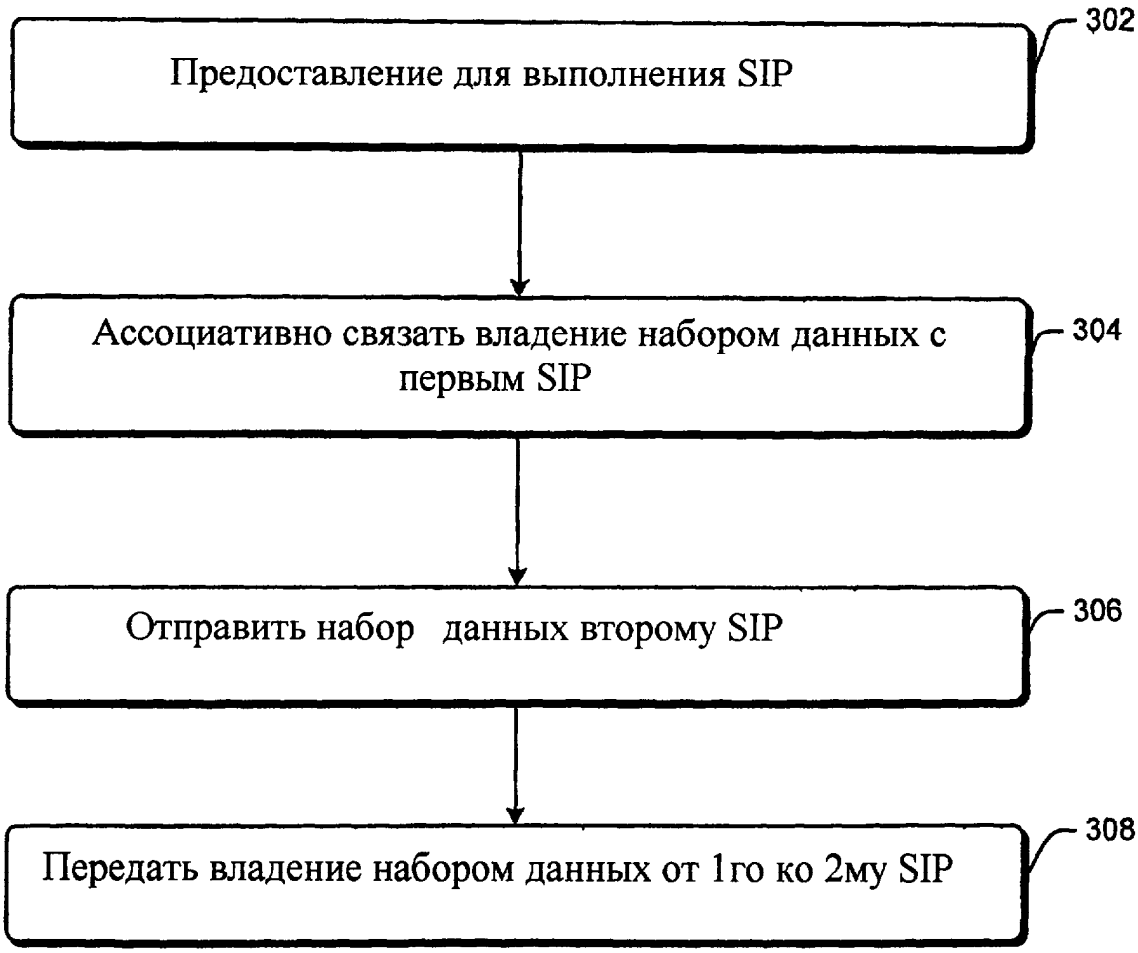
17. Носитель по п.13, в котором способ дополнительно содержит этапы:

определение, до посылки данных, пытаются ли посылающий изолированный программный процесс и принимающий изолированный программный процесс обратиться к блоку памяти в совместно используемом неупорядоченном массиве обмена, которым владеет другой изолированный программный процесс, и

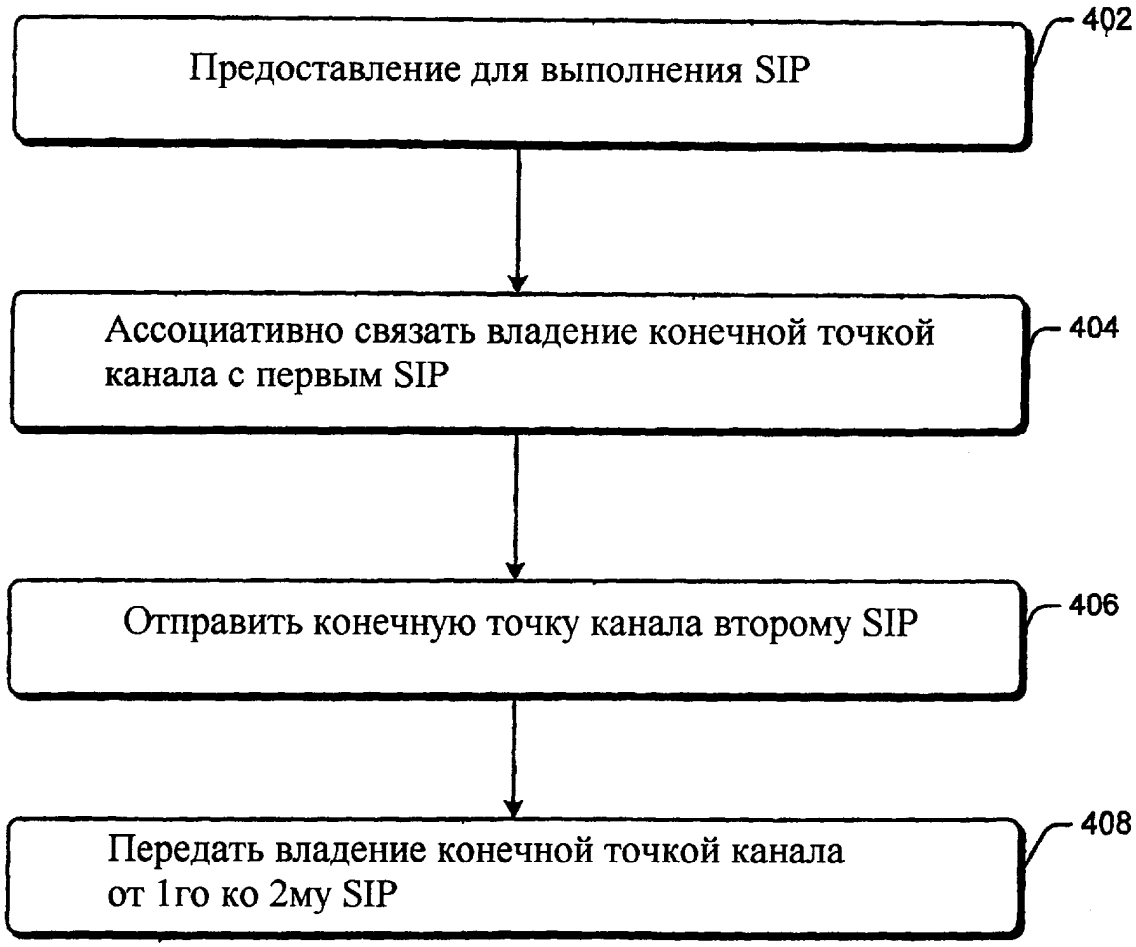
в ответ на определение предоставление индикации о том, пытаются ли посылающий изолированный программный процесс и принимающий изолированный программный процесс обратиться к блоку памяти в совместно используемом неупорядоченном массиве обмена, которым владеет другой изолированный программный процесс.


Память
Неупорядоченный массив обмена
290
210
230
240
250
Процесс
Процесс
Процесс
282
280
284
262
272
274
270
232
260
242
252
264
Координатор
Ядро 220
222
212
Операционная система
202
200

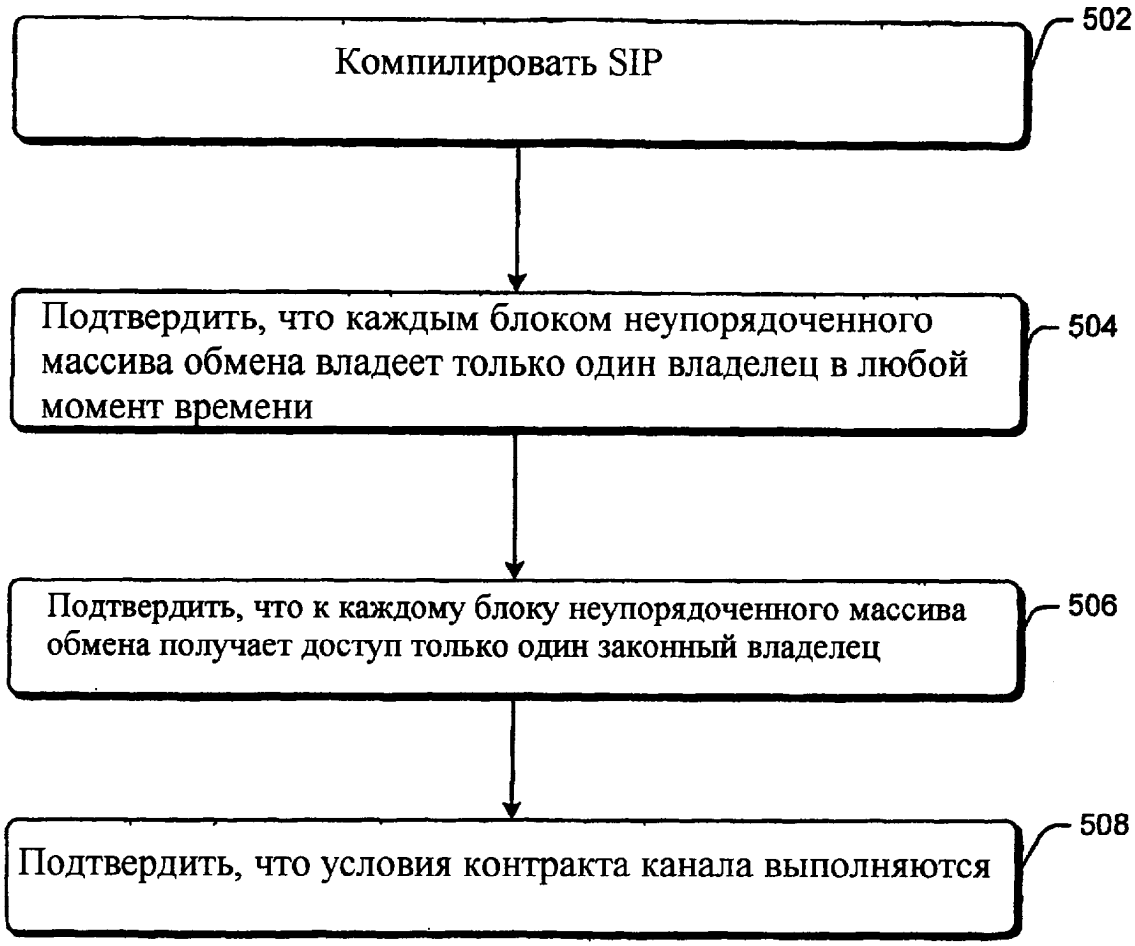
Фиг. 2


Предоставление для выполнения SIP
302
Ассоциативно связать владение набором данных с первым SIP
304
Отправить набор данных второму SIP
306
Передать владение набором данных от 1го ко 2му SIP
308
300

Фиг. 3


Предоставление для выполнения SIP
402
Ассоциативно связать владение конечной точкой канала с первым SIP
404
Отправить конечную точку канала второму SIP
406
Передать владение конечной точкой канала от 1го ко 2му SIP
408
400

Фиг. 4


Компилировать SIP
502
Подтвердить, что каждым блоком неупорядоченного массива обмена владеет только один владелец в любой момент времени
504
Подтвердить, что к каждому блоку неупорядоченного массива обмена получает доступ только один законный владелец
506
Подтвердить, что условия контракта канала выполняются
508
500

Фиг. 5