РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(19) **RU** (11) **2 507 569** (13) **C1**

(51) МПК
G06F 15/16 (2006.01)
H04L 29/02 (2006.01)

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21)(22) Заявка: **2012129031/08, 11.07.2012**

(24) Дата начала отсчета срока действия патента:
11.07.2012

Приоритет(ы):
(22) Дата подачи заявки: **11.07.2012**

(45) Опубликовано: **20.02.2014** Бюл. № **5**

(56) Список документов, цитированных в отчете о поиске: **US 2002107903 A1, 08.08.2002. US 2011145276 A1, 16.06.2011. US 6434145 B1, 13.08.2002. RU 2042193 C1, 20.08.1995. BY 5350 C1, 30.09.2003.**

Адрес для переписки:
127287, Москва, Старый Петровско-Разумовский пр-д, 1/23, стр.1, ОАО "Информационные технологии и коммуникационные системы"

(72) Автор(ы):
Морозов Вячеслав Викторович (RU),
Тычина Леонид Анатольевич (RU)

(73) Патентообладатель(и):
Открытое акционерное общество "Информационные технологии и коммуникационные системы" (RU)

(54) СПОСОБ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ УПОРЯДОЧЕННЫХ ПОТОКОВ ДАННЫХ

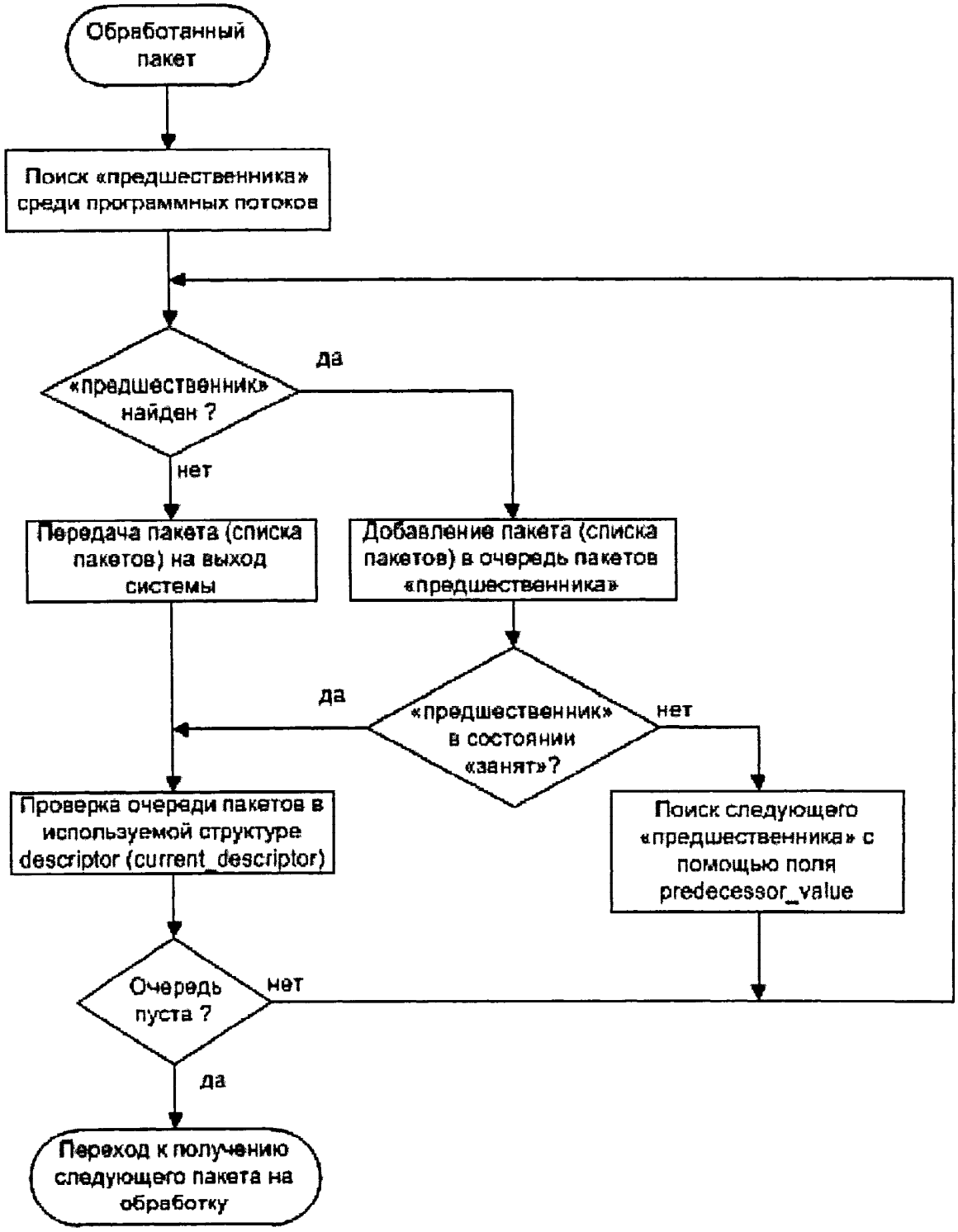
(57) Реферат:

Изобретение относится к вычислительной технике и может быть использовано для параллельной обработки нескольких цифровых потоков данных, каждый из которых представляет последовательность дискретных наборов данных определенного вида. Техническим результатом является повышение производительности обработки входных потоков за счет устранения ожидания момента окончания обработки очередной части входного потока в случаях, когда предыдущие части уже обработаны. Способ заключается в том, что получают входные потоки данных, передают части входных потоков данных для обработки в процессорные блоки, каждая часть каждого входного потока данных снабжается атрибутами - идентификатором входного потока и идентификатором положения данной части во входном потоке, обрабатывают части входных потоков данных, обеспечивают порядок следования частей выходных потоков данных, который соответствует порядку частей входных потоков данных, для этого проводят поиск процессорного блока, в котором обрабатывается часть определенного входного потока данных, находившаяся в определенном первом потоке перед частью, уже обработанной в рассматриваемом процессорном блоке, причем, если после поиска таких процессорных блоков найдено несколько, то выбирают тот процессорный блок, в котором обрабатывается часть определенного входного потока данных, расположенная наиболее близко к обработанной части определенного входного потока; передают обработанную часть определенного входного потока данных из рассматриваемого процессорного блока в выбранный процессорный блок, а также, при наличии, ранее полученные от других процессорных блоков обработанные части входного потока данных; если после поиска таких процессорных блоков не найдено, то передают обработанные части входного потока данных в соответствующий выходной

RU 2507569 C1

RU 2507569 C1

поток данных, в которых порядок следования частей соответствует порядку следования частей в соответствующем входном потоке, с учетом ранее полученных от других процессорных блоков обработанных частей входного потока данных. 10 ил., 1 табл.



Фиг. 2

RU 2507569 C1

RU 2507569 C1

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY

(19) **RU** (11) **2 507 569** (13) **C1**

(51) Int. Cl.
G06F 15/16 (2006.01)
H04L 29/02 (2006.01)

(12) ABSTRACT OF INVENTION

(21)(22) Application: **2012129031/08, 11.07.2012**

(24) Effective date for property rights:
11.07.2012

Priority:
(22) Date of filing: **11.07.2012**

(45) Date of publication: **20.02.2014 Bull. 5**

Mail address:
127287, Moskva, Staryj Petrovsko-Razumovskij pr-d, 1/23, str.1, OAO "Informatsionnye tekhnologii i kommunikatsionnye sistemy"

(72) Inventor(s):
Morozov Vjacheslav Viktorovich (RU),
Tychina Leonid Anatol'evich (RU)

(73) Proprietor(s):
Otkrytoe aktsionernoe obshchestvo "Informatsionnye tekhnologii i kommunikatsionnye sistemy" (RU)

RU 2507569 C1

(54) METHOD FOR PARALLEL PROCESSING OF ORDERED DATA STREAMS

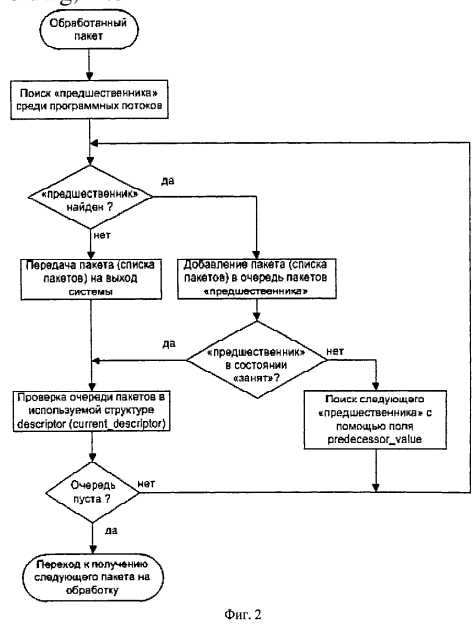
(57) Abstract:

FIELD: information technology.

SUBSTANCE: method involves obtaining input data streams; transmitting part of the input data streams for processing to processor units, each part of the input data streams being provided with attributes - an input stream identifier and an identifier of the position of that part in the input stream; processing parts of the input data streams; providing the sequence order of the parts of the input data streams which corresponds to the order of the parts of the input data streams, carried out by searching for a processor unit where part of the determined input data stream is processed, said part being in a certain first stream before the part already processed in the processor unit under consideration, wherein if a few of such processor units are found after search, the processor unit selected is that which processes part of the determined input data stream located closest to the processed part of the determined input stream; the processed part of the determined input data stream is transmitted from the considered processor unit to the selected processor unit, also in the presence of processed parts of the input data stream received from other processor units; if no such processor units are found after search, the processed parts of the input data stream are transmitted to the corresponding output data stream, where the sequence order of the parts corresponds to the sequence order of parts in the corresponding input stream, taking into account processed parts of the input data stream received from other processor units.

EFFECT: high efficiency of processing input streams by eliminating waiting for the end of processing the next part of the input stream in cases when previous parts have already been processed.

10 dwg, 1 tbl



Фиг. 2

RU 2507569 C1

Область техники, к которой относится изобретение

Предлагаемое изобретение относится к вычислительной технике и, в частности, к способам параллельной обработки нескольких цифровых потоков данных, каждый из которых представляет последовательность дискретных наборов данных определенного вида, например, IP пакетов и т.п.

Уровень техники

При обработке цифровых данных часто возникает задача параллельной обработки нескольких цифровых потоков данных, имеющих в общем случае разную скорость, с помощью автоматического многоканального устройства, обеспечивающего необходимую обработку каждого входного потока и передачу его в обработанном виде в соответствующий выходной поток, причем характерная скорость обработки данных в каждом канале обработки может быть существенно меньше скорости входного потока, а требуемая скорость обработки каждого входного потока без задержки обеспечивается за счет наличия множества каналов обработки.

Важными условиями успешной работы такого устройства является обеспечение точного соблюдения последовательности обработанных данных в каждом выходном потоке соответствующей последовательности во входном потоке, а также высокая производительность.

Характер обработки данных может быть различным, например это может быть преобразование входных пакетов протокола ATM в выходные пакеты протокола IP, преобразование входных зашифрованных/незашифрованных пакетов IP соответственно в расшифрованные/зашифрованные пакеты IP и т.д.

Так, известен способ передачи данных между одним или несколькими первыми сетевыми портами, принимающими один или несколько первых потоков данных, и одним или несколькими вторыми сетевыми портами, передающими один или несколько вторых потоков данных [1].

Способ включает следующие действия:

- посылку данных от одного или несколько первых потоков данных во множество каналов обработки;
- обработку данных параллельно двумя или более каналами обработки;
- получение данных, обработанных каналами обработки и
- посылку обработанных данных в один или несколько вторых потоков в один или несколько вторых портов,

причем, по крайней мере, в одном потоке из первого и второго потоков данные передаются в кадрах (frames), и каждый кадр в таком одном потоке обработан одним (единственным) каналом обработки, но, по крайней мере, два кадра в одном потоке обработаны двумя различными каналами обработки.

Данные, полученные в каждом первом потоке данных, передаются в соответствующий второй поток данных в том же самом порядке, в котором эти данные были в первом потоке данных.

После получения из первого потока каждый кадр перед посылкой на обработку в канал обработки снабжается дополнительными атрибутами (данными), в состав которых включают, по крайней мере,

- номер кадра в первом потоке и
- идентификатор канала, в который посылается данный кадр.

Для обеспечения правильного порядка следования обработанных кадров в соответствующем втором потоке организуется стек памяти, сформированный по принципу "первым пришел - первым ушел" (first in - first out, FIFO), куда записываются

идентификаторы канала для тех каналов обработки, в которые посылаются кадры из первого потока.

Поэтому когда после получения из канала обработки обработанный кадр посылается во второй поток, то это делается в порядке следования идентификаторов каналов в стеке FIFO.

Следует отметить, что при описании известного способа используется оригинальная терминология, согласно которой, в частности, термин "кадр" подразумевает дискретный набор цифровых данных определенного формата, соответствующий какому-либо принятому протоколу (ATM, IP и др.).

Для реализации известного способа используется система, включающая:

- первый блок для посылки данных от одного или нескольких первых потоков данных к множеству каналов обработки, в котором, по крайней мере, в одном потоке из первого и второго потоков, данные передаются в кадрах, причем первый блок выполнен с возможностью посылать каждый кадр только в один из каналов обработки, а также послать, по крайней мере, два разных кадра в два различных канала обработки;

- множество каналов обработки, каждый из которых содержит отдельный процессор;

- второй блок для получения данных, обработанных каналами обработки, и посылки обработанных данных в один или несколько вторых потоков в один или несколько вторых портов;

- блок управления порядком для обеспечения второго блока идентификаторами каналов для каждого канала обработки, которому посылается кадр первым блоком,

причем второй блок выполнен с возможностью получать идентификаторы каналов от блока управления порядком в таком же самом порядке, в котором соответствующие кадры расположены, по крайней мере, в одном первом потоке,

при этом, когда второй блок получает идентификатор канала, то второй блок посылает кадр из соответствующего канала обработки во второй поток, так что кадры посылаются, по крайней мере, в один второй поток из каналов обработки в порядке, определяемом идентификаторами каналов.

В известном способе предусмотрена обработка кадров как постоянного, так и переменного размера. В общем случае даже при обработке кадров постоянного размера по заранее установленному алгоритму время обработки отдельных кадров может отличаться из-за различных факторов (неодинаковая скорость работы отдельных каналов, разное время обращения к памяти и пр.). Поэтому вполне может возникнуть ситуация, когда текущий кадр первого потока уже обработан в каком-либо канале обработки, но не может быть передан на выход системы, поскольку предыдущий кадр, за которым он следует, еще не обработан и не передан на выход во второй поток. В этой ситуации система дожидается окончания обработки и передачи на выход сначала предыдущего кадра, а затем текущего, чтобы обеспечить правильный порядок следования кадров.

Еще более заметно такие задержки могут проявиться при обработке кадров переменного размера. Такие задержки снижают производительность системы, что является недостатком известного способа.

Известен также способ обеспечения работы сетевой системы для параллельной обработки потоков данных [2], причем система содержит

- первый блок, обеспечивающий

- прием входных потоков данных из внешних сетевых соединений;

- разделение входных потоков данных на части;
- снабжение каждой части каждого входного потока данных атрибутами;
- посылку частей каждого входного потока данных в процессорные блоки для обработки;
- совокупность процессорных блоков, каждый из которых имеет в своем составе процессор и буфер памяти для хранения обработанных частей входных потоков данных и обеспечивает
- обработку по заданному алгоритму частей входных потоков данных;
- посылку обработанных частей входных потоков данных в соответствующие выходные потоки данных;
- хранение обработанных частей входных потоков данных в буфере памяти до момента выполнения условий посылки этих частей в соответствующий выходной поток данных;
- второй блок, обеспечивающий
- прием обработанных частей входных потоков данных;
- формирование и модификацию выходных очередей, содержащих (выходные маркеры обработки), причем количество выходных очередей соответствует количеству выходных потоков данных;
- передачу обработанных частей входных потоков данных в виде соответствующих выходных потоков данных во внешнюю сеть;

причем первый блок связан с совокупностью процессорных блоков и со вторым блоком, а процессорные блоки также связаны со вторым блоком.

Согласно способу в одном из предлагаемых вариантов реализации:

- получают входные потоки данных из сетевых соединений в первом блоке;
- задают требуемое соответствие между входными и выходными потоками данных;
- формируют во втором блоке выходные очереди потоков, количество которых соответствует количеству выходных потоков данных;
- формируют в каждом процессорном блоке выходные очереди процессорных блоков, количество которых соответствует количеству выходных потоков данных;
- посылают части входных потоков данных для обработки в процессорные блоки, причем каждая часть каждого входного потока данных снабжается атрибутами, в состав которых включается
- идентификатор процессорного блока, в который посылается данная часть входного потока;
- идентификатор входного потока;
- помещают идентификатор процессорного блока, в который послана очередная часть входного потока данных для обработки, в выходную очередь потока второго блока, которая соответствует заданному выходному потоку и содержит (выходной маркер обработки);
- обрабатывают части входных потоков данных в процессорных блоках для получения соответствующих частей выходных потоков данных;
- записывают идентификатор процессорного блока, в котором закончилась обработка части определенного входного потока данных, в выходную очередь этого процессорного блока, которая соответствует заданному выходному потоку;
- обеспечивают {порядок следования} частей выходных потоков данных из (процессорных блоков), который соответствует порядку частей входных потоков данных, причем для обеспечения {правильного порядка следования}
- сравнивают идентификатор процессорного блока, в котором закончилась

обработка части первого потока, с правильным очередным идентификатором процессорного блока в (выходном маркере обработки) и, если при сравнении идентификаторы не совпадают, то

- сохраняют обработанную часть первого потока в буфере памяти данного процессорного блока;

- записывают идентификатор процессорного блока в выходную очередь данного процессорного блока;

- обрабатывают следующую часть входного потока данных в данном процессорном блоке;

- если при сравнении идентификаторы совпадают, то

- посылают части выходных потоков данных из процессорных блоков во второй блок для формирования выходных потоков данных, в которых порядок следования частей соответствует порядку следования частей в соответствующих входных потоках и

- после посылки очередной обработанной части первого потока модифицируют в каждом процессорном блоке идентификатор этого процессорного блока в выходной очереди этого процессорного блока для соответствующего выходного потока и в (выходном маркере обработки) соответствующего выходного потока. В известном способе предусмотрена обработка частей входного потока данных (сетевых пакетов) как постоянного, так и переменного размера.

Здесь также при обработке частей входного потока данных по заранее установленному алгоритму время обработки отдельных частей может отличаться из-за различных факторов (неодинаковая скорость работы отдельных процессорных блоков, разное время обращения к памяти и пр.). Поэтому вполне может возникнуть ситуация, когда отдельная часть входного потока уже обработана в каком-либо процессорном блоке, но не может быть немедленно передана на выход системы, поскольку предыдущая часть входного потока еще не обработана.

Для обеспечения порядка следования частей выходного потока данных, точно совпадающего с порядком следования частей соответствующего входного потока данных, используется специально сформированная очередь, первый элемент которой (выходной маркер обработки) является идентификатором процессорного блока, из которого должна поступить в выходной поток данных очередная обработанная часть входного потока данных.

В качестве идентификатора могут использоваться целые числа, адреса в памяти, индексы массивов и пр.

После посылки части входных потоков данных для обработки в процессорные блоки помещают идентификатор процессорного блока, в который послана очередная часть входного потока данных для обработки, в выходную очередь потока второго блока, которая соответствует заданному выходному потоку и содержит (выходной маркер обработки), причем

- перед записью идентификатора процессорного блока блокируют доступ к выходной очереди, обеспечивая, таким образом, исключительный доступ на запись со стороны этого процессорного блока (и запрет на запись со стороны любого другого процессорного блока);

- осуществляют запись идентификатора путем выполнения атомарных операций, а затем

- снимают блокировку доступа.

После окончания обработки в каком-либо процессорном блоке части входного

потока данных проводится проверка соответствия идентификатора этого процессорного блока и правильного идентификатора из выходного маркера обработки.

Если при сравнении эти идентификаторы совпадают, то обработанная часть входного потока данных передается на выход системы во второй блок, а выходной маркер обработки модифицируется путем удаления из его очереди номера данного процессорного блока.

Если же эти номера не совпадают, то обработанную часть первого потока сохраняют в буфере памяти данного процессорного блока и запоминают идентификатор данного процессорного блока в выходной очереди данного процессорного блока, организованной в формате стека памяти FIFO.

После этого процессорный блок останавливается и непрерывно производит проверку номера этого процессорного блока и правильного номера из выходного маркера обработки до момента, пока эти номера совпадут.

Согласно предпочтительному варианту реализации способа, если номера не совпадают, процессорный блок получает из первого блока новую часть входного потока данных и обрабатывает ее. После окончания обработки новой части входного потока данных снова проводится проверка соответствия идентификатора из стека выходной очереди этого процессорного блока и правильного номера из выходного маркера обработки.

Если эти номера совпадают, то обработанная часть входного потока данных передается из буфера памяти данного процессорного блока на выход системы во второй блок, а выходной маркер обработки модифицируется путем удаления из его очереди номера данного процессорного блока.

Модифицируется также стек выходной очереди данного процессорного блока путем удаления из него идентификатора данного процессорного блока, передавшего на выход обработанную часть входного потока данных.

Известный способ принят за прототип для предлагаемого технического решения.

Недостатком известного способа является невысокая производительность даже в предпочтительном варианте реализации способа, которая возникает из-за задержки при проверке идентификатора определенного процессорного блока и правильного номера из выходного маркера обработки до момента, пока эти номера совпадут, поскольку, если совпадение не произошло, то следующий момент проверки возникает только после обработки новой части входного потока данных.

Раскрытие изобретения

Техническим результатом является повышение производительности обработки входных потоков за счет устранения ожидания момента окончания обработки очередной части входного потока в случаях, когда предыдущие части уже обработаны.

Для этого предлагается способ параллельной обработки упорядоченных потоков данных в вычислительной системе, причем система содержит

- первый блок, обеспечивающий
- прием входных потоков данных из внешних сетевых соединений;
- разделение входных потоков данных на части;
- снабжение каждой части каждого входного потока данных атрибутами;
- передачу частей каждого входного потока данных в процессорные блоки для обработки;
- совокупность процессорных блоков, каждый из которых имеет в своем составе процессор и средства для хранения обработанных частей входных потоков данных и

обеспечивает

- обработку по заданному алгоритму частей входных потоков данных;
- передачу обработанных частей входных потоков данных в соответствующие выходные потоки данных;
- хранение обработанных частей входных потоков данных до момента выполнения условий посылки этих частей в соответствующий выходной поток данных;
- передачу обработанных частей входных потоков данных в другие процессорные блоки;
- получение обработанных частей входных потоков данных из других процессорных блоков;
- поиск заданных элементов в атрибутах частей входных потоков данных;

причем первый блок связан с совокупностью процессорных блоков,

способ, заключающийся в том, что

- получают входные потоки данных из сетевых соединений в первом блоке;
- передают части входных потоков данных для обработки в процессорные блоки, причем каждая часть каждого входного потока данных снабжается атрибутами, в состав которых включается
- идентификатор входного потока;
- идентификатор положения данной части во входном потоке;
- обрабатывают части входных потоков данных в процессорных блоках для получения соответствующих частей выходных потоков данных;
- обеспечивают порядок следования частей выходных потоков данных из процессорных блоков, который соответствует порядку частей входных потоков данных, причем для обеспечения порядка следования
- проводят поиск процессорного блока, в котором обрабатывается часть определенного входного потока данных, находившаяся в определенном первом потоке перед частью, уже обработанной в рассматриваемом процессорном блоке, причем,

если после поиска таких процессорных блоков найдено несколько, то

- выбирают тот процессорный блок, в котором обрабатывается часть определенного входного потока данных, расположенная наиболее близко к обработанной части определенного входного потока;
- передают обработанную часть определенного входного потока данных из рассматриваемого процессорного блока в выбранный процессорный блок, а также, при наличии, ранее полученные от других процессорных блоков обработанные части входного потока данных;

если после поиска таких процессорных блоков не найдено, то

- передают обработанные части входного потока данных в соответствующий выходной поток данных, в которых порядок следования частей соответствует порядку следования частей в соответствующем входном потоке, с учетом ранее полученных от других процессорных блоков обработанных частей входного потока данных.

Таким образом, в отличие от известного способа в предложенном способе после окончания обработки части определенного входного потока данных не происходит задержки, а обработанная часть входного потока данных передается из рассматриваемого процессорного блока в выбранный процессорный блок. Вслед за передачей обработанной части рассматриваемый процессорный блок сразу принимает для обработки очередную часть какого-либо входного потока данных и начинает ее обрабатывать.

Можно отметить также, что эта очередная часть может принадлежать другому входному потоку данных, отличному от определенного входного потока данных, к которому принадлежала ранее обработанная и переданная часть. Такая возможность отсутствует в прототипе, поскольку в нем используются идентификаторы процессорных блоков, привязанные, в конечном счете, к какому-либо одному входному потоку данных до окончания его обработки, а это также снижает производительность системы.

Краткое описание чертежей

На фиг.1 показана схема, поясняющая принцип организации очереди FIFO с помощью атрибутов Next и Last структуры данных packets_info.

На фиг.2 показан общий алгоритм передачи обработанного пакета на выход системы.

На фиг.3 показан алгоритм поиска "предшественника" среди обрабатываемых пакетов.

На фиг.4 показан алгоритм передачи списка обработанных пакетов программным потоком своему текущему "предшественнику".

На фиг.5 показан алгоритм проверки очереди пакетов, полученных от "последователей", которые программный поток должен передать на выход системы.

На фиг.6 показан алгоритм поиска "предшественника" с помощью поля predecessor_value, выполняемый программным потоком при организации отправки обработанных пакетов на выход системы.

На фиг.7 показан жизненный цикл объекта, доступ к которому контролируется с помощью механизма подсчета ссылок.

На фиг.8 показан алгоритм функции GetReference механизма подсчета ссылок.

На фиг.9 показан алгоритм функции RequestToDisableSharedMode механизма подсчета ссылок.

На фиг.10 показан алгоритм функции ReleaseReference механизма подсчета ссылок.

Осуществление изобретения

Рассмотрим пример реализации предложенного способа в сетевом маршрутизаторе, выполненном в виде многопроцессорной вычислительной системы и предназначенном для преобразования множества входных потоков данных, получаемых из внешней сети передачи данных (например, сетевых пакетов данных из сети Интернет), во множество выходных потоков данных, передаваемых, например, во внутреннюю корпоративную сеть передачи данных.

Для определенности можно рассмотреть входные потоки данных, представляющие собой последовательность пакетов данных, сформированных по протоколу TCP/IP и зашифрованных с использованием какого-либо стандарта (например, DES) с известными параметрами.

Задачей маршрутизатора будет расшифровка входных потоков данных и передача их во внутреннюю сеть передачи данных потребителям (обычным пользователям).

Для приема входных потоков данных маршрутизатор содержит

- нескольких сетевых интерфейсов, обеспечивающих прием и передачу сетевых пакетов данных;

- нескольких процессорных блоков, каждый из которых представляет собой процессор общего назначения (например, на базе архитектуры x86 или ARM), выполняющих обработку принятых пакетов,

- оперативной памяти, предназначенной для хранения принятых сетевых пакетов, а также информации, необходимой для функционирования системы.

Архитектура процессоров общего назначения должна поддерживать следующие типы операций:

- атомарная (непрерываемая) операция чтения ячейки памяти с последующей записью нового значения (далее обозначается AtomicExchange), например, для архитектуры x86 - это команда процессора "xchg";

- атомарная (непрерываемая) операции чтения ячейки памяти с последующей записью считанного значения увеличенного на заданное число (далее - AtomicAdd), например, для архитектуры x86 - это команда процессора "lock xchgadd".

Маршрутизатор функционирует под управлением операционной системы (ОС), способной функционировать в многопроцессорной конфигурации (например, ОС Linux).

Для реализации предложенного способа для каждого процессорного блока обеспечивается выполнение дополнительно следующих функций, отсутствующих в прототипе:

- передачу обработанных частей входных потоков данных в другие процессорные блоки;

- получение обработанных частей входных потоков данных из других процессорных блоков;

- поиск заданных элементов в атрибутах частей входных потоков данных;

Передача обработанных частей входных потоков данных в соответствующие выходные потоки данных обеспечивается каждым процессорным блоком. Непосредственная передача данных во внутреннюю сеть может осуществляться с помощью одной или нескольких сетевых карт, соединенных с внутренней сетью.

Функции, требуемые для реализации предложенного способа, должны обеспечиваться прикладным программным обеспечением (ППО), которое может быть разработано специалистом по программированию (программистом) на основе приведенных сведений о назначении функций.

Компоненты ОС, обеспечивающие управление сетевыми интерфейсами - драйверы сетевых интерфейсов - помещают принятые сетевые пакеты в оперативную память. Дополнительно к каждому принятому пакету драйвер создает в оперативной памяти специальную структуру данных (далее - packet_info), которая состоит из следующих полей (атрибутов):

- адрес пакета данных из определенного входного потока данных в оперативной памяти;

- адрес следующего пакета данных из определенного входного потока данных в очереди (последовательности) пакетов (далее Next);

- адрес последнего пакета данных в очереди (последовательности) пакетов (далее Last);

Атрибуты Last и Next предназначены для организации очереди из принятых пакетов, формируемой по схеме FIFO. У последнего пакета данных в очереди в поле Next записан 0 (нуль).

Принцип организации очереди FIFO с помощью указанных полей приведен на фиг.1.

После копирования пакета в память драйвер добавляет соответствующую ему структуру packet_info в FIFO очередь принятых пакетов. Доступ к очереди синхронизируется посредством стандартной функции ОС (примитива синхронизации, например, spinlock в ОС Linux), обеспечивающей эксклюзивный доступ к синхронизируемому объекту (например, ячейке памяти). Указанный примитив функционирует следующим образом: для получения доступа к объекту компонент ОС

должен "захватить" (lock) примитив, после чего может модифицировать объект, а затем "отпустить" примитив (unlock).

Обработка пакетов выполняется в программных потоках обработки ОС (например, kernel threads в ОС Linux), причем количество программных потоков не превышает количества процессорных блоков в системе, и каждый программный поток выполняется только в одном процессорном блоке.

Каждый программный поток имеет два состояния:

- "занят" - производит обработку сетевого пакета, а также действия по сохранению порядка следования пакетов;
- "свободен" - ожидает нового пакета для обработки, при этом поток не выполняется своим процессорным блоком.

Ожидание программным потоком нового пакета для обработки может быть реализовано с помощью стандартного механизма синхронизации ОС, например ожидание на примитиве синхронизации семафор в ОС Linux.

Количество программных потоков, находящихся в определенном состоянии, хранится в оперативной памяти в виде специальной структуры данных (далее - threads_info). Доступ к этой структуре также синхронизирован с помощью примитива синхронизации очереди FIFO.

Драйвер сетевого интерфейса после добавления packet_info в очередь принятых пакетов на основе данных структуры threads_info определяет, есть ли в данный момент программный поток обработки в состоянии «свободен». Если такой программный поток есть, то драйвер использует соответствующий механизм примитива синхронизации ОС для активации программного потока.

Для обработки сетевого пакета в программном потоке используют специальную структуру (далее - descriptor), которая хранит всю необходимую информацию для выполнения требуемых действий над сетевым пакетом и отправки его на выход из системы.

Каждый программный поток имеет свой фиксированный набор структур descriptor, зарезервированных в оперативной памяти. Количество структур в наборе каждого программного потока равно количеству программных потоков в системе. Этим гарантируется, что у программного потока всегда будет свободная структура descriptor, для обработки нового пакета.

Структура descriptor может быть в трех состояниях:

- "занят" - используется для обработки пакета;
- "свободен" - может использоваться программным потоком для обработки нового пакета;
- переходное состояние от "занят к "свободен" - пакет обработан, но структура еще не может быть использована для обработки нового пакета.

Управление состояниями структуры descriptor осуществляется посредством специального программного механизма подсчета ссылок на объект и дополнительных флагов, описанных ниже.

Структура descriptor состоит из полей, приведенных в табл.1

Таблица 1

Обозначение	Тип	Назначение
state	ссылка	характеризует состояние структуры descriptor
id	число	идентификатор входного сетевого потока, из которого взят пакета, описываемый данной структурой descriptor

order_id	беззнаковое целое число	порядковый номер пакета во входном сетевом потоке; считается, что пакет А расположен в входном сетевом потоке перед пакетом Б, если разность их порядковых номеров, представленная в виде целого числа со знаком, будет отрицательна
predecessor_value	адрес	адрес структуры descriptor, которая описывает пакет ("предшественник"), расположенный во входном сетевом потоке данных перед обрабатываемым пакетом
predecessor_state	ссылка	ссылка для синхронизации доступа к полю predecessor_value
packets_list	адрес	адрес первого элемента очереди, состоящей из структур packet_info; данная очередь используется для передачи пакетов между программными потоками в процессе определения порядка следования их на выход системы; наряду с адресом packet_info в данном поле храниться два флага Predesessor_Refn Successor_Ref
stop	число	признак, что descriptor не находится в состоянии "занят"
free	число	признак, что descriptor находится в состоянии "свободен"

Структура descriptor переходит в состояние "свободен", только когда ссылка state перешла в состояние модификации объекта, таким образом, в состоянии "свободен" ссылка state находится в состоянии модификации.

В состоянии "занят" структуры descriptor ссылка state находится в состоянии совместного доступа к объекту.

Получение сетевого пакета программным потоком для обработки

Программный поток после активации драйвером или окончания обработки пакета получает доступ к очереди принятых пакетов, "захватывая" примитив синхронизации, и забирает первый пакет из очереди.

Если нет пакетов для обработки, то программный поток освобождает примитив синхронизации, переходит в состояние «свободен» и ожидает активации на соответствующем примитиве синхронизации ОС.

Если пакет имеется, то после извлечения его из очереди программный поток формирует идентификатор входного сетевого потока данных, которому принадлежит пакет, используя номер сетевого интерфейса в ОС, тип протокола сетевого уровня, а также информацию из заголовка сетевого уровня (например, информацию из IP заголовка: IP адреса источника и получателя, тип протокола транспортного уровня). Затем пакету присваивается порядковый номер, который отражает его расположение во входном потоке данных. Идентификатор потока и порядковый номер сохраняются программным потоком в оперативной памяти в структуре descriptor, адрес которой записан в переменной free_descriptor. У каждого программного потока имеется своя переменная free_descriptor. В данной переменной всегда в момент получения пакета на обработку хранится адрес структуры descriptor в состоянии "свободен" из набора структур descriptor программного потока.

После заполнения соответствующих полей программный поток переводит структуру descriptor в состояние "занят" (ссылка state в режим совместного использования). Затем записывает адрес структуры descriptor в оперативную память в переменную current_descriptor. С каждым программным потоком связана своя переменная current_descriptor.

Затем программный поток освобождает примитив синхронизации очереди, после чего считается, что он перешел в состояние "занят". Программный поток приступает к обработке пакета (расшифровка по заданному алгоритму, возможны также дополнительные предусмотренные действия по маршрутизации, применению правил фильтрации и т.п.).

Передача пакета на выход системы

После обработки для определения правильного порядка следования пакета на выход из системы (на соответствующий сетевой интерфейс) программный поток выполняет поиск "предшественника" (predecessor) из обрабатываемых в данный

момент другими программными потоками пакетов.

"Предшественник" - это пакет из того же входного сетевого потока данных, что и обработанный данным программным потоком, но расположенный во входном сетевом потоке перед ним, т.е. имеющий меньший порядковый номер.

В структуре descriptor предусмотрено поле predecessor_value, в которое помещается адрес структуры descriptor, используемой программным потоком, производящим обработку найденного "предшественника". Доступ остальных программных потоков к полю predecessor_value синхронизируется посредством механизма подсчета ссылок. Для этой цели в descriptor предусмотрено поле predecessor_state типа reference.

Передача пакета на выход из системы зависит от наличия "предшественника".

Если "предшественник" не был найден, то это означает, что все пакеты из определенного входного сетевого потока до текущего обработанного пакета уже переданы на выход системы, вследствие чего обработанный пакет передается на выход системы (драйверу сетевого интерфейса для отправки).

Если "предшественник был найден, то программный поток добавляет пакет (список пакетов в общем случае, см. ниже) в очередь пакетов packets_list в структуру данных descriptor "предшественника". В процессе добавления программный поток проверяет признаки состояния в структуре descriptor "предшественника". Если признаки указывают состояние "занят", то пакет успешно добавлен в его очередь. Теперь передачу пакета далее на выход системы выполнит программный поток, выполняющий обработку "предшественника".

Если "предшественник" находится в переходном состоянии (в состоянии "занят" он быть не может, поскольку программный поток, добавляющий пакет (список пакетов), удерживает ссылку на него), то добавления не происходит. При этом, если у "предшественника" были пакеты в очереди, то программный поток формирует из этих пакетов и своего пакета (списка пакетов) новый список пакетов.

Затем программный поток проводит поиск "предшественника" в состоянии "занят", используя поле predecessor_value в структуре descriptor "предшественника". Если поиск успешен, то найденный "предшественник" используется для передачи ему пакетов аналогично описанному выше. Если же "предшественник" так и не был найден, то программный поток передает список пакетов на выход системы (драйверам соответствующих сетевых интерфейсов).

После передачи списка пакетов на выход системы (согласно описанному выше алгоритму) программный поток проверяет очередь пакетов в своей структуре descriptor (переменная current_descriptor). Если в процессе проверки программный поток обнаруживает, что очередь пуста либо в данный момент другой программный поток пытается добавить пакеты для отправки, то данный программный поток выставляет признак переходного состояния своей структуры descriptor и переходит к получению следующего пакета. Иначе программный поток получает все пакеты из очереди и передает их по описанному выше алгоритму.

Схема передача пакетов на выход системы приведена на фиг.2.

Поиск "предшественника" среди обрабатываемых пакетов

Программный поток выполняет поиск "предшественника", просматривая поочередно переменные current_descriptor остальных программных потоков (далее - descr) и сравнивая значения идентификатора входного сетевого потока и порядкового номера обрабатываемого пакета с такими же параметрами своего current_descriptor (далее - current).

Перед выполнением сравнения программный поток получает ссылку de-scr.state, что блокирует переход этой структуры descriptor в свободное состояние (для повторного использования).

Если descr - первый найденный описатель, удовлетворяющий требованиям "предшественника", то программный поток пытается получить его ссылку prede-cessor_state. В случае успеха адрес descr сохраняется как "кандидат в предшественники" (переменная pred). Полученная ссылка descr.predecessor_state гарантирует, что в состоянии перехода descr из состояния "занят" в состояние "свободен" значение descr.predecessor_value сохраниться до тех пор, пока current удерживает эту ссылку (descr не может перейти в состояние "свободен", поскольку программный поток ранее получил его ссылку descr.state).

Если descr - первая найденная структура descriptor, удовлетворяющая вышеперечисленным требованиям, то программный поток получает ссылку descr.predecessor_state и, в случае успеха, descr сохраняется как "кандидат в предшественники".

Если на предыдущих итерация поиска уже был найден «кандидат в предшественники», то в таком случае производится сравнение порядковых номеров descr и "кандидата в предшественники" для определения, кто из них "ближе" к current.

Если ближе оказался descr, то программный поток получает ссылку descr.predecessor_state и в случае успеха запоминает descr в качестве "кандидата в предшественники", а ссылки бывшего "кандидата в предшественники" освобождаются.

В результате успешного поиска последний найденный "кандидат в предшественники" будет считаться "предшественником", его адрес будет сохранен в current_descriptor.predecessor_value, а программный поток передаст ему свой обработанный пакет, чтобы программный поток "предшественника" отправил его на выход системы после своего пакета.

Последовательность поиска "предшественника" среди обрабатываемых пакетов показана на фиг.3.

Передача пакетов "предшественнику"

Для передачи пакета "предшественнику" используются поля соответствующей структуры packet_info для организации очереди пакетов. Для этой цели в структуре descriptor предусмотрено поле packets_list, которое хранит адрес первой структуры packet_info в очереди (первый элемент очереди).

У "предшественника" проверяется поле stop структуры descriptor. Если оно равно 1 (установлено), то это означает, что данная структура descriptor переходит из состояния "занят" в состояние "свободен" и, следовательно, уже не может использоваться как "предшественник", который передаст обработанный пакет. В этом случае необходимо выполнить новый поиск предшественника путем последовательного просмотра списка "предшественников", взяв за начало списка поле pre-decessor_value текущего "предшественника".

Для передачи пакета (списка пакетов в общем случае) "предшественнику" программный поток записывает с помощью AtomicExchange флаг Successor_ref в поле packets_list "предшественника". Затем поток формирует новую очередь из своего пакета (списка пакетов) и очереди, хранившейся в packets_list "предшественника".

Если программный поток обнаруживает, что в packets_list «предшественника» был установлен флаг Predecessor_ref, т.е. "предшественник" в данный момент проверяет

свое поле packets_list на наличие новых пакетов, то программный поток считает его в состоянии перехода от "занят" к "свободен".

Если флаг Predecessor_ref не был установлен, то программный поток записывает адрес первой структуры packets_info сформированной очереди в packets_list "предшественника", используя AtomicExchange. Если к этому моменту флаг Predecessor_ref уже был установлен (анализируется значение packets_list, которое вернула функция AtomicExchange), то программный поток считает, что «предшественник» в переходном состоянии.

Если флаг не был установлен, то это признак того, что пакет (список пакетов) успешно передан "предшественнику" и программный поток может перейти к проверке своего поля packets_list для получения новых пакетов для передачи на выход системы от своих "последователей".

Если программный поток не смог добавить пакет (список пакетов) в очередь ("предшественник" в переходном состоянии), то он записывает в поле stop "предшественника" значение 1 и выполняет поиск "предшественника", используя поле predecessor_value текущего "предшественника" pred.

Схема передачи списка обработанных пакетов программным потоком своему текущему "предшественнику" показана на фиг.4.

Получение пакетов от "последователей" для передачи на выход системы

Поток с помощью функции AtomicAdd устанавливает флаг Predecessor_ref в current_descriptor.packets_list.

Если в значении, которое вернула функция AtomicAdd, в качестве адреса начала очереди (первого элемента) записано значение 0, то очередь пуста, и структура current_descriptor теперь находится в "переходном" состоянии (для "последователей" об этом сигнализирует установленный функцией AtomicAdd флаг Prede-cessor_ref).

Если функция AtomicAdd вернула значение, в котором был установлен флаг Successor_ref, то в данный момент "последователь" добавляет пакеты в очередь current_descriptor. Соответственно, "последователь" обнаружит флаг Predecessor_ref, когда будет добавлять новый список пакетов. Таким образом, «последователь» определит, что структура current_descriptor находится в переходном состоянии, и будет искать другого "предшественника".

Если флаг Successor_ref не был установлен и очередь не пуста, то программный поток из возвращенного значения извлекает адрес первого элемента pack-et_info очереди, после чего записывает (посредством вызова функции AtomicExchange) значение 0 в packet_list описателя.

Если к этому моменту "последователь" приступил к добавлению своих заданий, то в возвращенном значении функции AtomicExchange будет установлен флаг Successor_ref. В этом случае также считается, что current_descriptor перешел в переходное состояние.

Если флаг Successor_ref не был установлен, то программный поток выполнил проверку очереди и может выполнять передачу полученных пакетов на выход системы, используя "предшественника" (как описано выше), либо, в случае отсутствия "предшественника", самостоятельно передает пакеты на соответствующие выходные сетевые интерфейсы (используя программный интерфейс ОС для передачи пакетов).

Если в результате проверки очереди пакетов структура current_descriptor перешла в переходное состояние, то программный поток переводит ссылки predeces-sor_state и state в состояние "запрос на эксклюзивное использование" (с помощью вызова функции RequestToDisableSharedMode для каждой из них), после чего программный

поток может перейти к выбору следующего пакета для обработки, используя другую свободную структуру descriptor из своего набора.

При переходе ссылки predecessor_state в режим модификации в результате освобождения всех ссылок проверяется поле predecessor_value: если его значение не нулевое, то освобождаются его (predecessor_value) ссылки predecessor_state и state.

При переходе ссылки state в режим модификации в результате освобождения всех ссылок выполняются следующие действия в структуре descriptor:

- выполняется функция EnableSharedMode для ссылки predecessor_state;
- записывается значение 0 в packets_list;
- записывается значение 0 в поле stop;
- записывается значение 1 в поле free - признак, что структура descriptor в состоянии "свободен".

После окончания обработки пакета (current_descriptor в переходном состоянии) поток выполняет поиск descriptor в свободном состоянии путем анализа поля free у всех дескрипторов из набора программного потока.

Последовательность проверки очереди пакетов, полученных от "последователей", которые программный поток должен передать на выход системы, показана на фиг.5.

Поиск "предшественника" с помощью поля predecessor value

Программный поток получает адрес следующего "предшественника" (pred.pdecessor_value) у своего "предшественника" (pred). Если адрес не нулевой и выполняются следующие условия: значение pred.stop не равно 1 и программный поток успешно получил ссылки pred.predecessor_state и pred.state, то данная структура descriptor становится новым "предшественником" для потока и поиск прекращается.

Если условие не выполнено, то производится следующая итерация поиска, где выполняются аналогичные действия для структуры, адрес которой записан в поле predecessor_value только что проверенной структуры descriptor. Количество итераций поиска (размер цепочки "предшественников") ограничено количеством потоков, уменьшенном па 1 (текущий поток).

После завершения поиска предыдущий "предшественник" освобождается: разблокируются его ссылки predecessor_state и state соответственно.

Если в результате поиска "предшественник" не найден, то программный поток самостоятельно передаст пакеты на соответствующие выходные сетевые интерфейсы.

Схема поиска "предшественника" с помощью поля predecessor_value, выполняемого программным потоком при организации отправки обработанных пакетов на выход системы, показана на фиг.6.

Поиск структуры descriptor, находящейся в состоянии "свободен"

После перехода current_descriptor в переходное состояние (программный поток закончил все действия по обработке и передаче пакетов, как обработанного самостоятельно, так и полученных от «последователей»), программный поток выполняет поиск свободной структуры descriptor, которая будет использоваться для обработки нового пакета.

Поиск выполняется путем проверки поля free у всех структур descriptor в наборе программного потока. Поскольку количество структур в наборе соответствует количеству программных потоков в системе и текущий программный поток не блокирует никаких структур descriptor в других программных потоках, то, как минимум, одна структура descriptor будет в состояние "свободен". Данное утверждение следует из описанных выше алгоритмов поиска "предшественника" и механизма

подсчета ссылок, приведенного далее.

Для реализации предложенного способа используется ряд вспомогательных процедур, описанных ниже.

Механизм подсчета ссылок на объект

Жизненный цикл какого-либо объекта в системе (в качестве объекта может, например, рассматриваться переменная в оперативной памяти), использующего механизм подсчета ссылок (далее - механизм), начинается с его инициализации (соответствующих данных механизма, в том числе как части объекта). На этой стадии доступ к объекту имеет только его создатель (или владелец, в качестве которого могут выступать функции и программные модули), остальные программные элементы системы (функции и программные модули) не имеют никаких сведений о существовании объекта (объект "не опубликован"). После инициализации владелец объекта выполняет необходимые действия для "публикации" объекта - с этого момента остальные программные элементы системы могут пытаться получить доступ к его содержимому.

Для разрешения доступа к объекту владелец вызывает функцию Enable-SharedMode, и объект переходит в состояние совместного использования. Теперь программные элементы системы, желающие получить доступ к объекту, должны получить ссылку, используя функцию GetReference рассматриваемого механизма. В случае успешного получения ссылки гарантируется, что объект не перейдет в состояние модификации до момента освобождения ссылки. Таким образом, в этом состоянии гарантируется т.н. доступ к объекту только на чтение (read-only). По окончании использования программные элементы системы должны вызвать функцию ReleaseReference механизма, чтобы освободить ссылку.

Для выхода из режима совместного использования один из программных элементов системы должен вызвать функцию RequestToDisableSharedMode, в результате действия которой объект переходит в состояние ожидания окончания совместного использования. На этом этапе получение новых ссылок невозможно - т.е. GetReference будет завершаться неуспешно.

После освобождения последней ссылки объект перейдет в состоянии эксклюзивного доступа, при этом освободивший последнюю ссылку программный элемент системы может модифицировать объект. Для перехода обратно в состояние совместного использования программный элемент системы должен заново вызвать функцию EnableSharedMode.

Для уничтожения объекта должна быть устранена "публикация" объекта, в результате чего гарантируется, что ни один программный элемент системы теперь не сможет обратиться к объекту для получения ссылки. После этого вызывается функция RequestToDisableSharedMode. Впоследствии программный элемент системы, освободивший последнюю ссылку, может удалить объект.

Данные, необходимые для функционирования механизма подсчета ссылок, представлены в виде структуры reference.

На фиг.7 показан жизненный цикл объекта, доступ к которому контролируется с помощью механизма подсчета ссылок.

Реализация механизма подсчета ссылок

Приведенный выше механизм подсчета ссылок может быть реализован с помощью указанных выше функций AtomicExchange и AtomicAdd.

Для реализации приведенного выше механизма необходима структура данных, состоящая из следующих полей:

- переменная reference, которая совмещает счетчик обращений к объекту для получения ссылки (функция GetReference) с флагом Re-quest_To_Release, который, в случае если он установлен, означает, что запрещено получение новых ссылок на объект. Шаг счетчика обращений (ссылок) - константа Reference;

- переменная release_ref, являющаяся счетчиком выполненных операций освобождения ссылок, используется при переходе к режиму эксклюзивного доступа к объекту;

- release_target, представляющая собой количество ссылок, которые должны быть освобождены для перехода к состоянию эксклюзивного доступа.

Инициализация объекта

В reference устанавливается флаг Request_To_Release. Таким образом, объект переводится в режим эксклюзивного доступа и может быть "опубликован".

Переход в состояние совместного использования (EnableSharedMode)

Переменным release_ref и release_target присваивается значение 0. Атомарно устанавливается счетчик обращений к объекту равным 0, и сбрасывается флаг Re-quest_To_Release, разрешая получение ссылок.

Получение ссылки

Для получения ссылки увеличивается счетчик reference на единицу Reference и проверяется его предыдущее значение. Если в reference не был установлен флаг Request_To_Release, то ссылка успешно получена и объект находится в состоянии совместного использования. Если флаг был установлен, то ссылка не получена и доступ к объекту не разрешен.

На фиг.8 показан алгоритм работы функции GetReference для механизма подсчета ссылок.

Переход к состоянию модификации

В reference устанавливается флаг Request_To_Release. Если значение счетчика было равно 0, то все ссылки уже были освобождены (или их вообще никто не получал), и объект переходит в состояние эксклюзивного доступа.

Если не все ссылки были освобождены к моменту установки флага Re-quest_To_Release, то в release_target записывается предыдущее значение счетчика reference, которое отражает количество неосвобожденных ссылок. Затем release ref увеличивается на единицу Reference, и полученное в результате этой операции предыдущее значение release_ref сравнивается с release_target. Если значения равны, то это означает, что с момента установки флага Request_To_Release было произведено освобождение всех ссылок и, соответственно, объект перешел в состояние эксклюзивного доступа.

На фиг.9 показан алгоритм работы функции RequestToDisableSharedMode для механизма подсчета ссылок.

Освобождение ссылки

Счетчик reference уменьшается на единицу Reference. Затем, если был установлен флаг Request_To_Release, увеличивается счетчик release_ref и сравнивается его предыдущее значение (которое вернула функция AtomicAdd) с release_target. В случае совпадения значений объект переходит в состояние эксклюзивного доступа (была освобождена последняя ссылка).

На фиг.10 показан алгоритм работы функции ReleaseReference для механизма подсчета ссылок.

Все описанные процедуры и алгоритмы могут быть реализованы в ППО специалистом по программированию (программистом) на основе известности

выполняемых функций.

Предложенный способ позволяет избежать задержек в работе процессорных блоков из-за устранения необходимости ожидания окончания обработки отдельных пакетов в других процессорных блоках.

Необходимо отметить, что возможны и другие варианты реализации предложенного способа, отличающиеся от описанного выше и зависящие от личных предпочтений при программировании отдельных действий и функций.

Источники информации

1. Патент США №6434145, Processing of network data by parallel processing channels, приоритет от 22.06.1998 г., МКИ H04L 12/56.

2. Заявка США №09/797197, Methods and systems for the order serialization of information in a network processing environment, публикация №20020107903, приоритет от 01.03.2001 г., МКИ G06F 15/16.

Формула изобретения

Способ параллельной обработки упорядоченных потоков данных в вычислительной системе, причем система содержит

первый блок, обеспечивающий

прием входных потоков данных (из внешних сетевых соединений);

разделение входных потоков данных на части;

снабжение каждой части каждого входного потока данных атрибутами;

передачу частей каждого входного потока данных в процессорные блоки для обработки;

совокупность процессорных блоков, каждый из которых имеет в своем составе процессор и средства для хранения обработанных частей входных потоков данных и обеспечивает

обработку по заданному алгоритму частей входных потоков данных;

передачу обработанных частей входных потоков данных в соответствующие выходные потоки данных;

хранение обработанных частей входных потоков данных до момента выполнения условий посылки этих частей в соответствующий выходной поток данных;

передачу обработанных частей входных потоков данных в другие процессорные блоки;

получение обработанных частей входных потоков данных из других процессорных блоков;

поиск заданных элементов в атрибутах частей входных потоков данных;

причем первый блок связан с совокупностью процессорных блоков,

способ, заключающийся в том, что

получают входные потоки данных из сетевых соединений в первом блоке;

передают части входных потоков данных для обработки в процессорные блоки, причем каждая часть каждого входного потока данных снабжается атрибутами, в состав которых включается идентификатор входного потока;

идентификатор положения данной части во входном потоке;

обрабатывают части входных потоков данных в процессорных блоках для получения соответствующих частей выходных потоков данных;

обеспечивают порядок следования частей выходных потоков данных из процессорных блоков, который соответствует порядку частей входных потоков данных, причем для обеспечения порядка следования

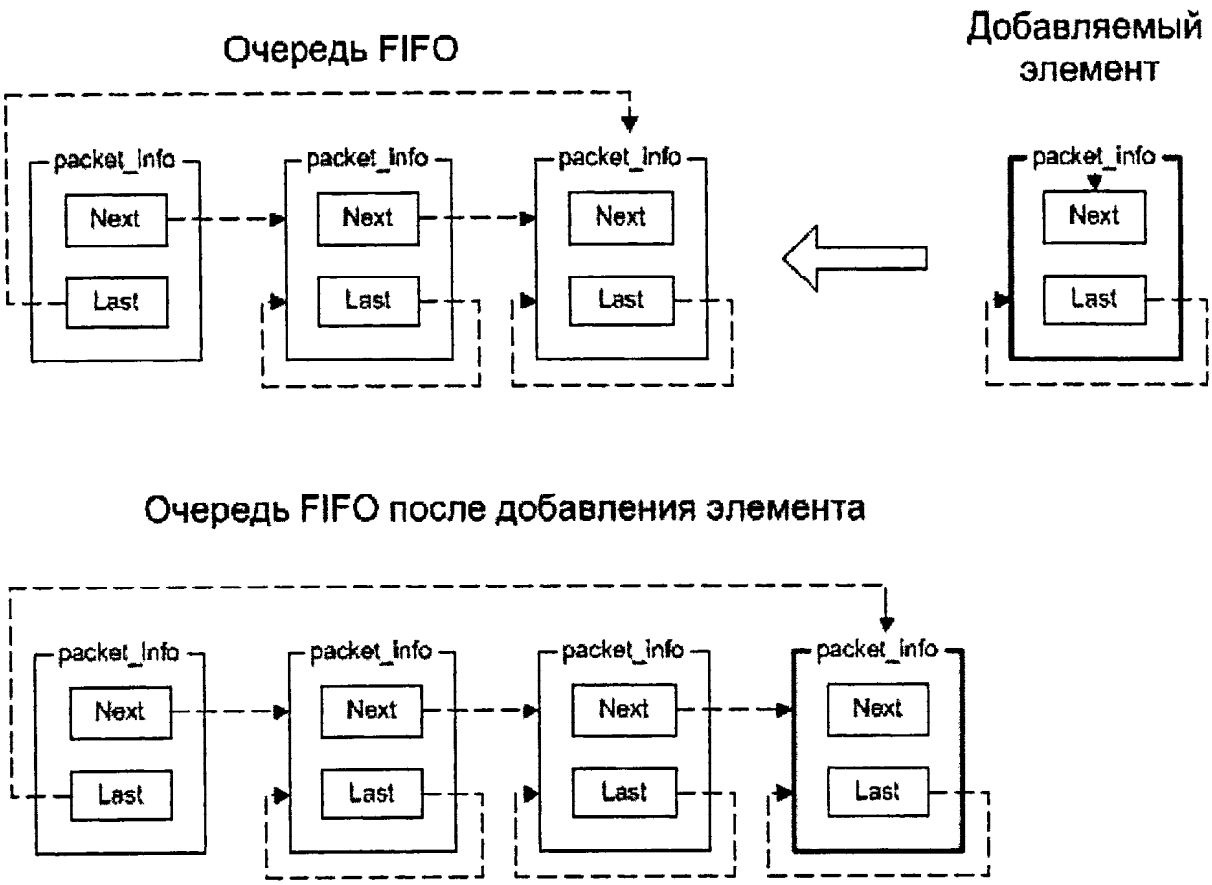
проводят поиск процессорного блока, в котором обрабатывается часть определенного входного потока данных, находившаяся в определенном первом потоке перед частью, уже обработанной в рассматриваемом процессорном блоке, причем,

если после поиска таких процессорных блоков найдено несколько, то выбирают тот процессорный блок, в котором обрабатывается часть определенного входного потока данных, расположенная наиболее близко к обработанной части определенного входного потока;

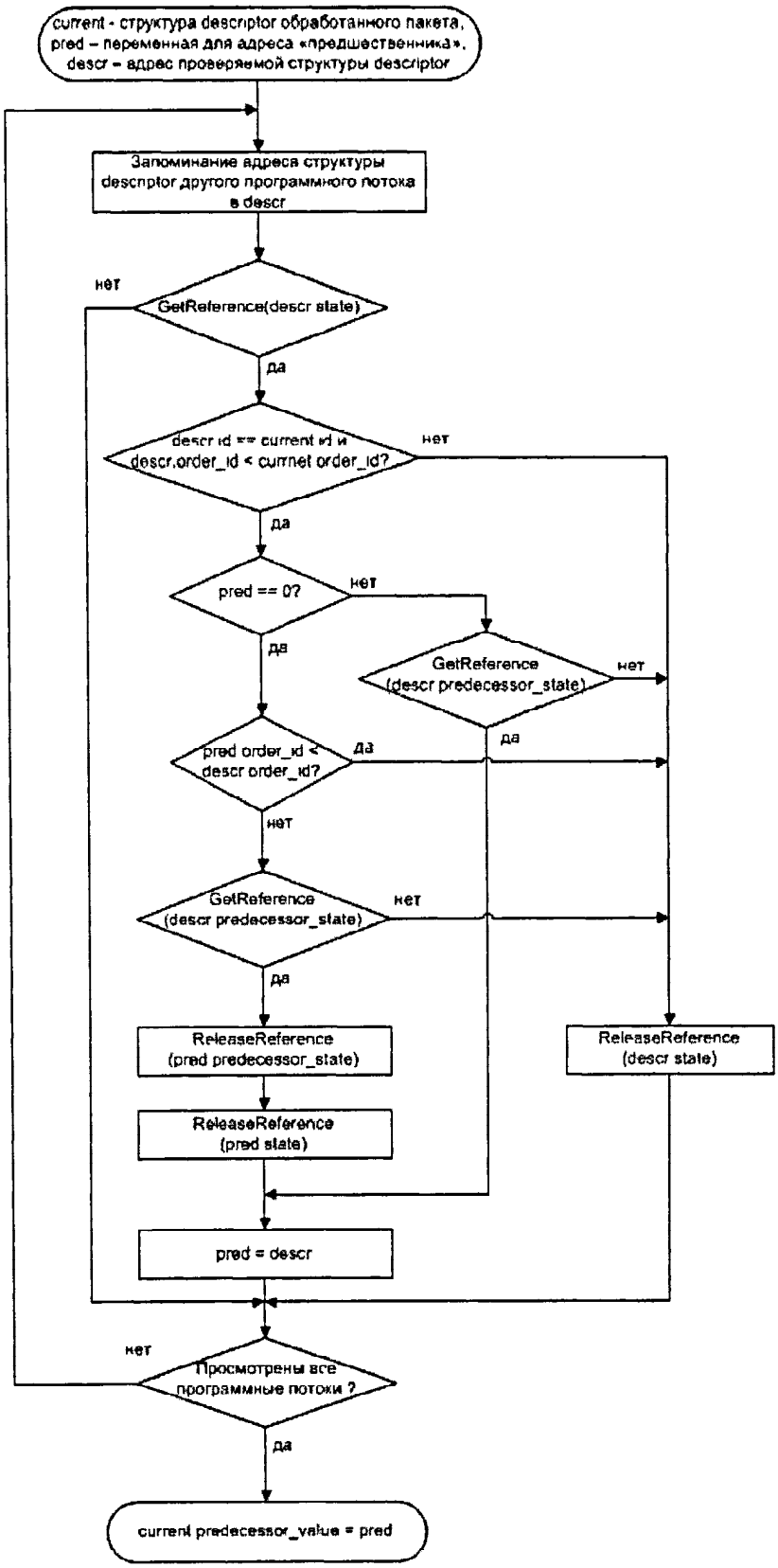
передают обработанную часть определенного входного потока данных из рассматриваемого процессорного блока в выбранный процессорный блок, а также, при наличии, ранее полученные от других процессорных блоков обработанные части входного потока данных;

если после поиска таких процессорных блоков не найдено, то

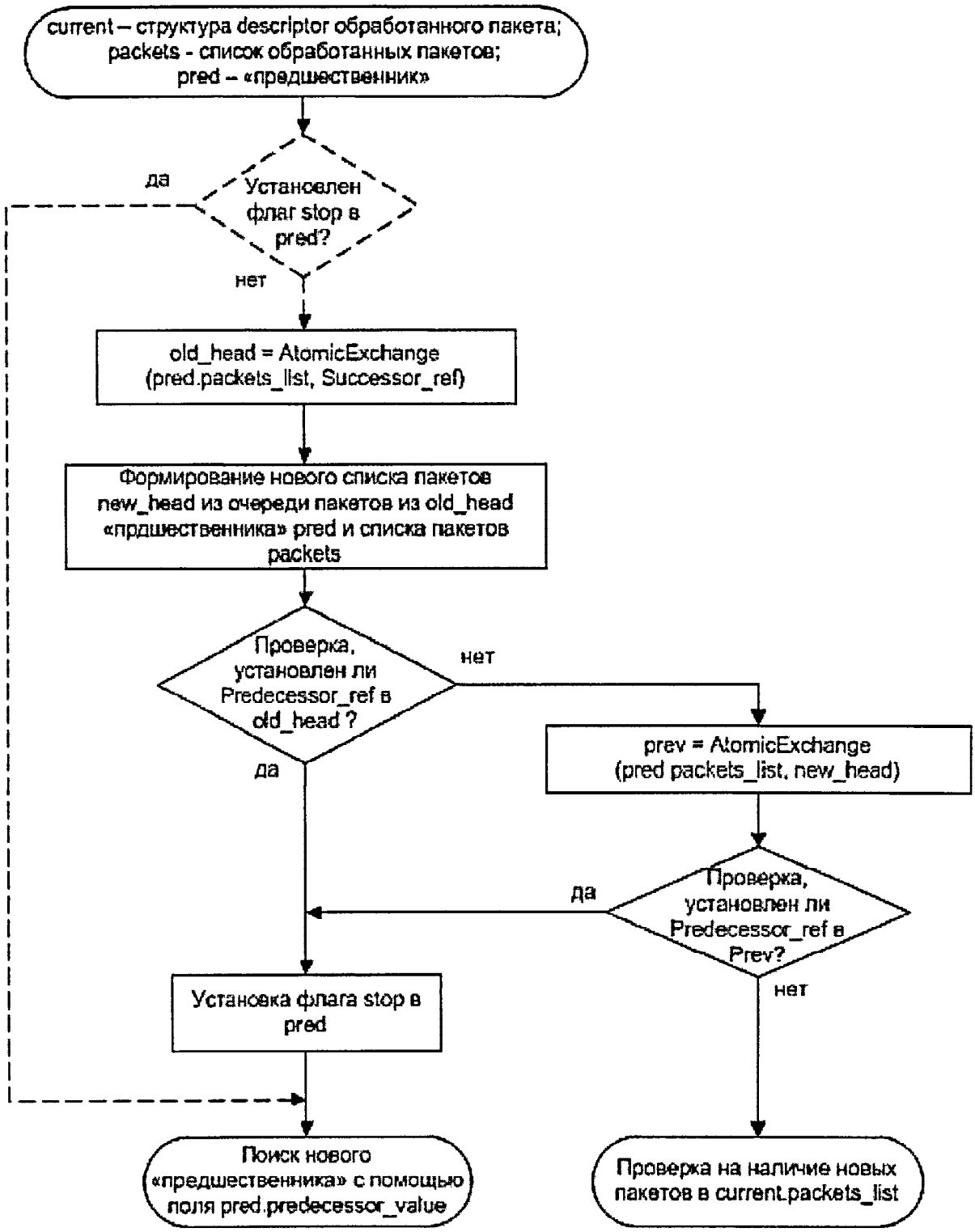
передают обработанные части входного потока данных в соответствующий выходной поток данных, в которых порядок следования частей соответствует порядку следования частей в соответствующем входном потоке, с учетом ранее полученных от других процессорных блоков обработанных частей входного потока данных.


Очередь FIFO
Добавляемый элемент
packet_info
Next
Last
packet_info
Next
Last
packet_info
Next
Last
packet_info
Next
Last
Очередь FIFO после добавления элемента
packet_info
Next
Last
packet_info
Next
Last
packet_info
Next
Last
packet_info
Next
Last

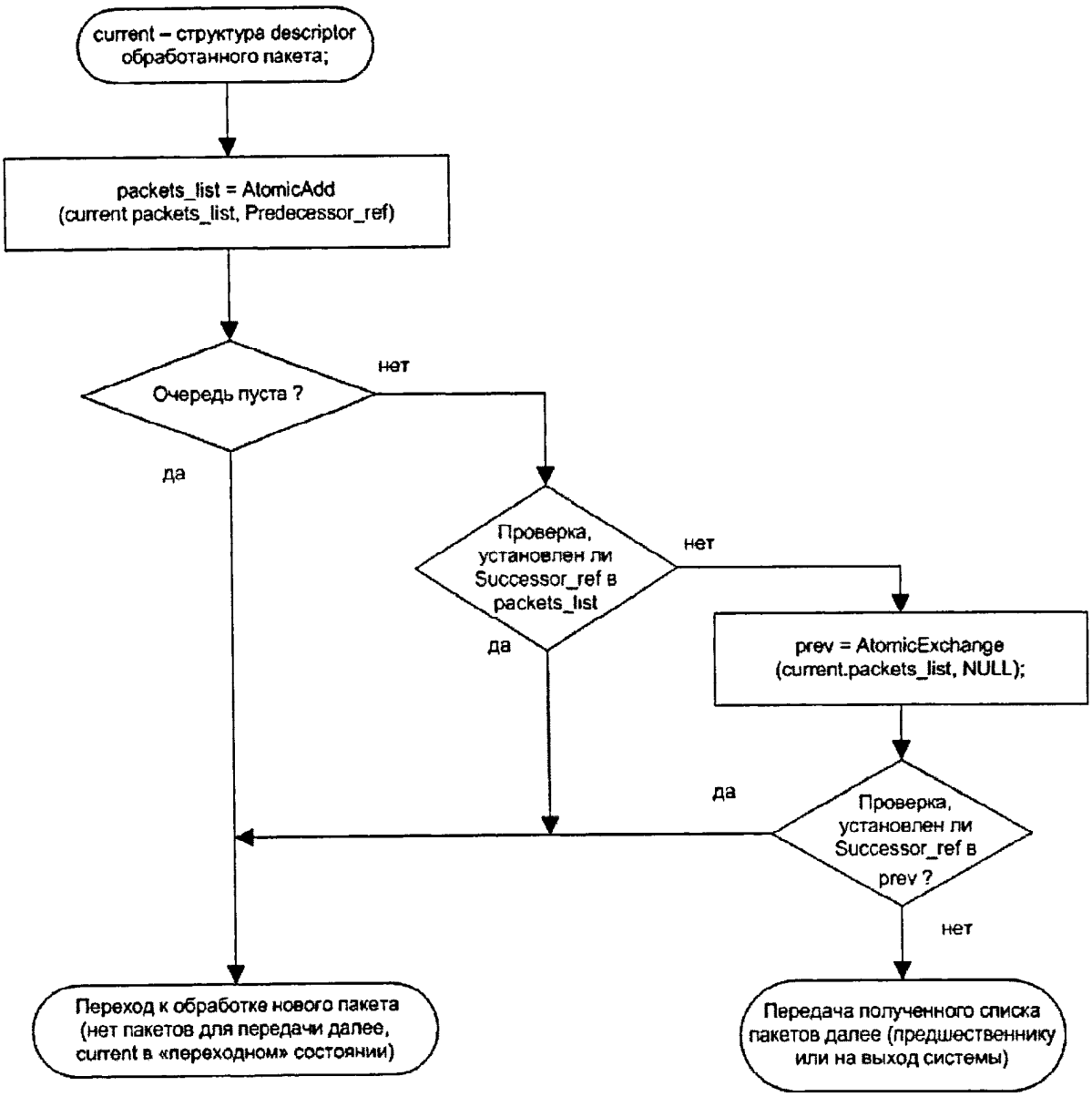
Фиг. 1


current - структура descriptor обработанного пакета,
pred – переменная для адреса «предшественника»,
descr – адрес проверяемой структуры descriptor
Запоминание адреса структуры descriptor другого программного потока в descr
нет
GetReference(descr state)
да
descr id == current id и descr.order_id < curmet order_id?
нет
да
pred == 0?
нет
да
GetReference (descr predecessor_state)
нет
да
pred order_id < descr order_id?
да
нет
GetReference (descr predecessor_state)
нет
да
ReleaseReference (pred predecessor_state)
ReleaseReference (pred state)
ReleaseReference (descr state)
pred = descr
нет
Просмотрены все программные потоки ?
да
current predecessor_value = pred

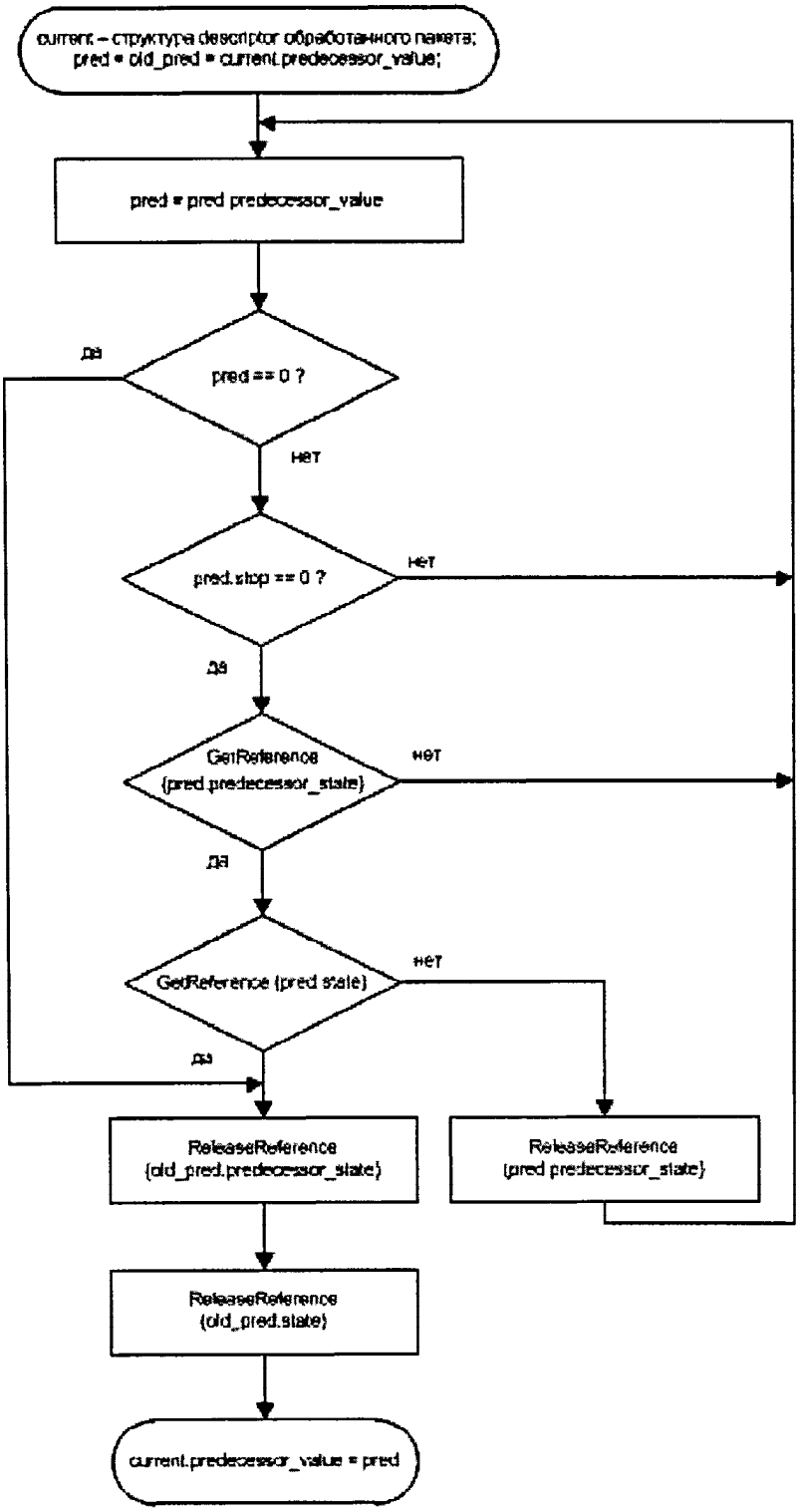
Фиг. 3


current – структура descriptor обработанного пакета;
packets - список обработанных пакетов;
pred – «предшественник»
Установлен флаг stop в pred?
да
нет
old_head = AtomicExchange (pred.packets_list, Successor_ref)
Формирование нового списка пакетов new_head из очереди пакетов из old_head «прдшественника» pred и списка пакетов packets
Проверка, установлен ли Predecessor_ref в old_head ?
нет
да
prev = AtomicExchange (pred packets_list, new_head)
Проверка, установлен ли Predecessor_ref в Prev?
да
нет
Установка флага stop в pred
Поиск нового «предшественника» с помощью поля pred.predecessor_value
Проверка на наличие новых пакетов в current.packets_list

Фиг. 4


current – структура descriptor обработанного пакета;
packets_list = AtomicAdd (current packets_list, Predecessor_ref)
Очередь пуста ?
нет
да
Проверка, установлен ли Successor_ref в packets_list
нет
да
prev = AtomicExchange (current.packets_list, NULL);
да
Проверка, установлен ли Successor_ref в prev ?
нет
Переход к обработке нового пакета (нет пакетов для передачи далее, current в «переходном» состоянии)
Передача полученного списка пакетов далее (предшественнику или на выход системы)

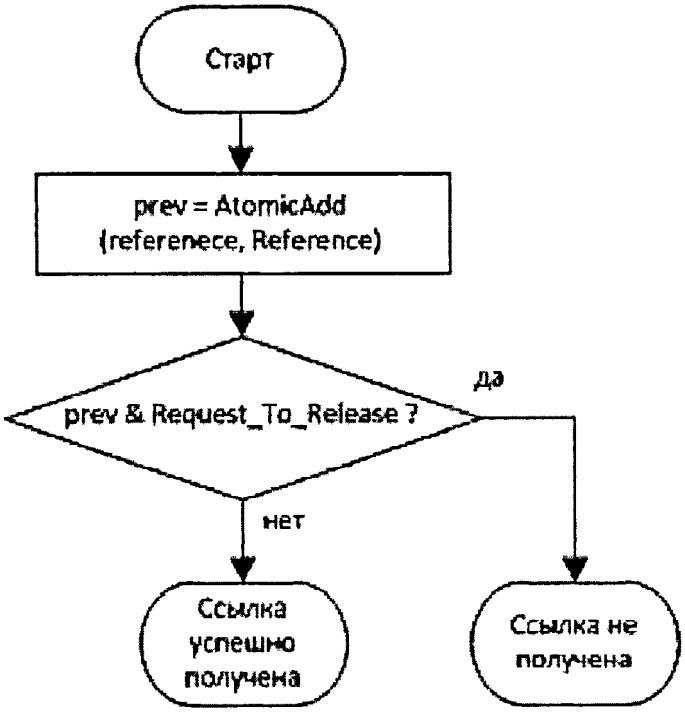
Фиг. 5


current – структура descriptor обработанного пакета;
pred = old_pred = current.predecessor_value;
pred = pred.predecessor_value
pred == 0 ?
да
нет
pred.stop == 0 ?
нет
да
GetReference (pred.predecessor_state)
нет
да
GetReference (pred.state)
нет
да
ReleaseReference (old_pred.predecessor_state)
ReleaseReference (pred.predecessor_state)
ReleaseReference (old_pred.state)
current.predecessor_value = pred

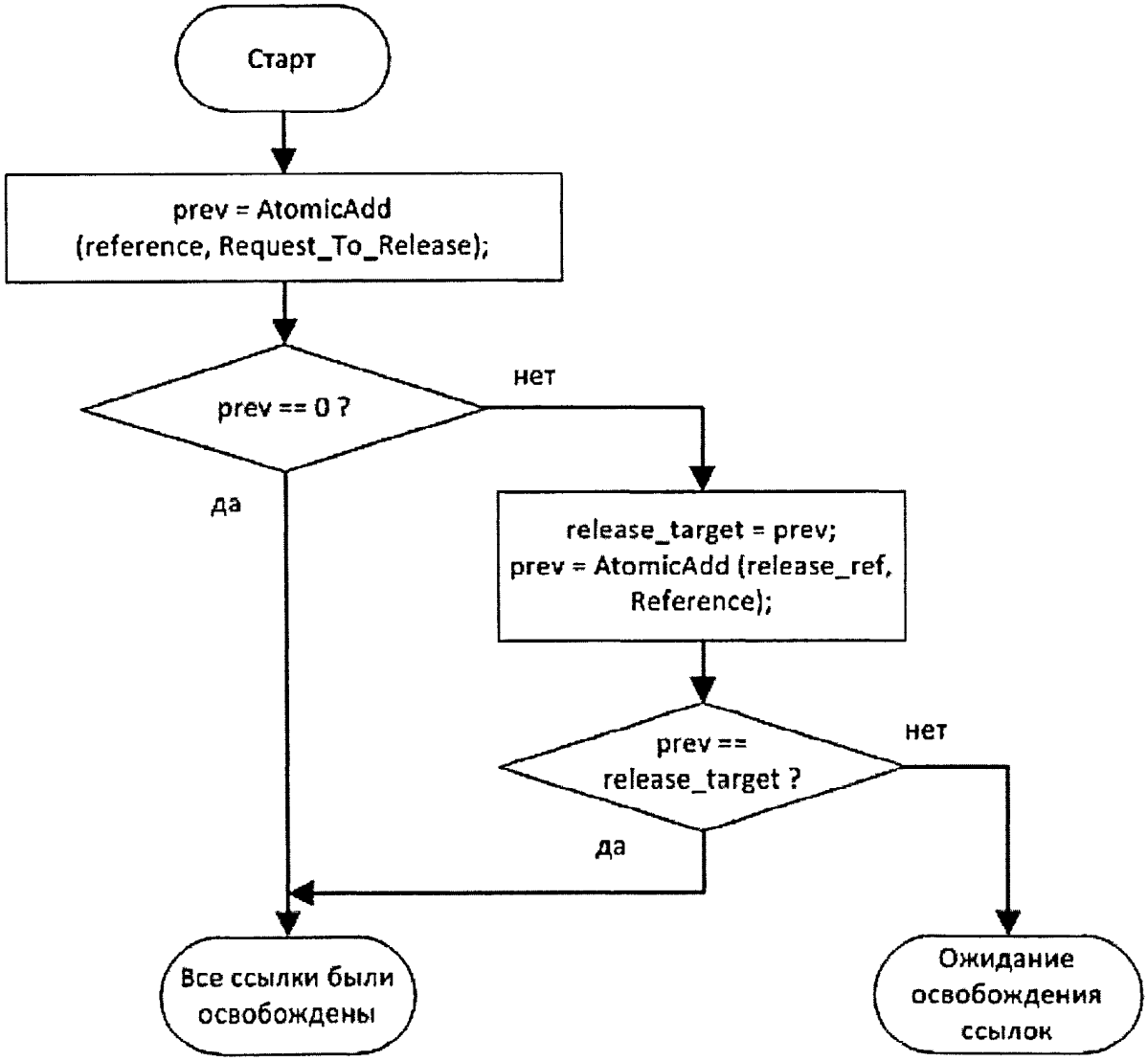
Фиг. 6


Создание/
инициализация
Публикация объекта
Модификация
Освобождена
последняя ссылка
Перевод в режим
совместного доступа
Ожидание
модификации
Запрос
модификации
Совместное
использование
Удаление публикации
Уничтожение

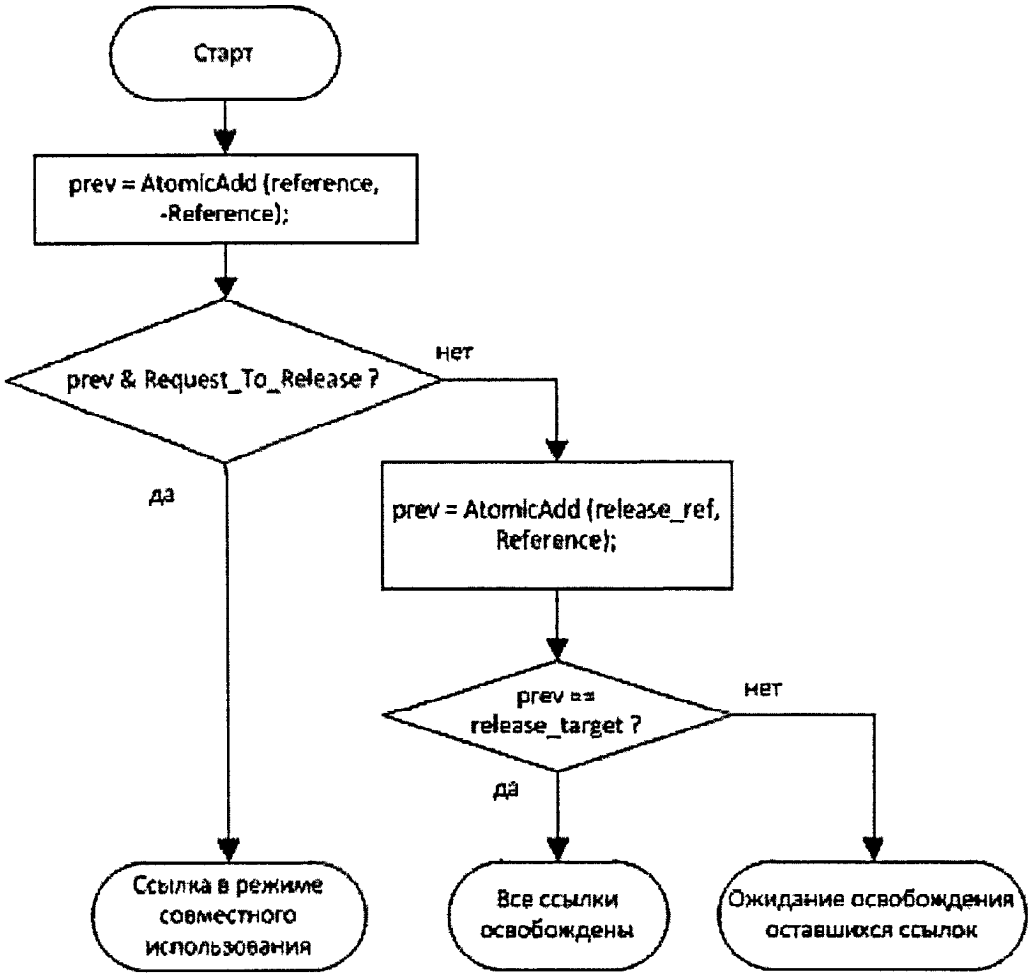
Фиг. 7


Старт
prev = AtomicAdd
(reference, Reference)
prev & Request_To_Release ?
да
нет
Ссылка
успешно
получена
Ссылка не
получена

Фиг. 8


Старт
prev = AtomicAdd
(reference, Request_To_Release);
prev == 0 ?
нет
да
release_target = prev;
prev = AtomicAdd (release_ref,
Reference);
prev ==
release_target ?
нет
да
Все ссылки были
освобождены
Ожидание
освобождения
ссылок

Фиг. 9


Старт
prev = AtomicAdd (reference, -Reference);
prev & Request_To_Release ?
нет
да
prev = AtomicAdd (release_ref, Reference);
prev == release_target ?
нет
да
Ссылка в режиме совместного использования
Все ссылки освобождены
Ожидание освобождения оставшихся ссылок

Фиг. 10