РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
АВТОМЕТРИЯ
№ 3 1996

УДК 681.3.06

Ю. И. Колосова
(Новосибирск)

ОБЩАЯ СХЕМА АЛГОРИТМОВ СОХРАНЕНИЯ ПРИЧИННОГО ПОРЯДКА СОБЫТИЙ И ЕЕ ИСПОЛЬЗОВАНИЕ

Существующие алгоритмы управления порядком приема сообщений отдельными процессами в соответствии с порядком их посылки из других процессов распределенной программы были специально разработаны для уменьшения недетерминизма, возникающего из-за асинхронной природы каналов связи. Рассматривается общая схема таких алгоритмов и предлагается новый алгоритм, созданный на ее основе. Показывается, что для некоторого класса задач он может быть более практичным. Демонстрируется пример использования такой схемы управления при разработке конкретной задачи.

Введение. Асинхронная природа каналов связи распределенной системы является основной причиной недетерминизма, проявляющегося, например, в том, что при нескольких выполнениях на одних и тех же исходных данных программа выдает разные результаты. Для снижения степени такого недетерминизма были разработаны алгоритмы, управляющие порядком приема сообщений отдельными процессами в соответствии с порядком их посылки из других процессов [1—3]. В них используется распространенная модель, где распределенная программа состоит из N последовательных процессов P_1, P_2, ..., P_N, взаимодействующих только посредством посылки сообщения по надежным каналам, чья задержка связи может быть произвольной, но конечной. В основе алгоритмов лежит понятие причинного порядка событий. Алгоритмы различаются структурой и объемом вспомогательных данных, но сходны схемой управления порядком взаимодействий процессов, которая может быть использована и при создании новых алгоритмов.

В следующих двух разделах вводится понятие причинного порядка событий и рассматривается общая схема управления его сохранением. Далее представляется новый алгоритм сохранения причинного порядка событий, анализируются объемы дополнительной информации, доказывается его корректность. Затем приводится пример решения конкретной задачи с использованием общей схемы управления.

1. **Причинный порядок событий.** С абстрактной точки зрения распределенное вычисление может быть описано типами и порядком событий, случившихся в каждом из N процессов. Различают три типа событий: посылка или прием сообщения и внутреннее событие, которое влияет только на локальное состояние процесса. Множество событий в процессе P_i будем обозначать как $\{e_{i1}, e_{i2}, e_{i3}, ...\}$ в соответствии с порядком их выполнения.

Причинный порядок событий в распределенной программе основан на известном отношении «случилось перед», обозначаемом «→» [4]. «Случилось перед» является транзитивным, нерефлексивным и асимметричным отно-

шением частичного порядка на множестве событий и определяется следующим образом:

$$\forall e_{ip},\ \forall e_{jq}\ \ e_{ip} \to e_{jq} \Leftrightarrow \begin{cases} (i = j) \wedge (p < q) \\ \text{либо существует сообщение } m \text{ такое, что} \\ e_{ip} \text{ есть посылка } m \text{ в } P_j, \\ e_{jq} \text{ есть прием } m \text{ от } P_i, \\ \text{либо существует событие } e_{ks} \text{ такое, что} \\ e_{ip} \to e_{ks} \to e_{jq}. \end{cases}$$

Выполнение события e_{jq} причинно зависит от события e_{ip}, если $e_{ip} \to e_{jq}$. Если же $\neg(e_{ip} \to e_{jq}) \wedge \neg(e_{jq} \to e_{ip})$, то $e_{ip} \parallel e_{jq}$.

Алгоритмы сохранения причинного порядка событий гарантируют, что если e_{ip} и e_{jq} (включая $i = j$) выполняют посылку сообщений процессу P_k, а e_{kx} и e_{ky} — прием соответствующих сообщений, то из $e_{ip} \to e_{jq}$ следует, что $e_{kx} \to \to e_{ky}$. Другими словами, если существует отношение «случилось перед» между событиями посылки сообщений одному и тому же процессу, то существует отношение «случилось перед» и между событиями их приема. Можно сказать, что прием таких сообщений обслуживается по дисциплине FSFR — «первым послан — первым принят». Для параллельно посылаемых сообщений прием обслуживается подобно известной дисциплине FIFO.

На рис. 1 показана пространственно-временная диаграмма хода выполнения трех процессов (представлен стрелками слева-направо) и их взаимодействий при посылке сообщения (показаны стрелками от процесса-источника к процессу-адресату). Процесс P_3 получает сообщение M_2 перед сообщением M_1, хотя событие посылки сообщения M_1 находится в отношении «случилось перед» с событием посылки сообщения M_2. Для устранения такого нарушения порядка следования событий и предназначены рассматриваемые алгоритмы.

2. Общая схема алгоритмов сохранения причинного порядка событий. В алгоритмах [1—3] используются похожие способы управления сохранением причинного порядка событий, общая схема которых представлена в этом разделе.

Алгоритмы представляют собой распределенную mail-систему, каждый блок mail_i которой управляет посылкой и приемом сообщений процесса P_i, обозначаемых $m.\text{v}$ и называемых основным значением. При этом используется классический метод дополнения для управления взаимодействием. Каждое сообщение m состоит из основного значения $m.\text{v}$ и заголовка $m.h$, содержащего сведения о всех известных к текущему моменту посылках. С каждым приемом связаны разрешающие условия encon_i (enabling conditions). Основное значение сообщения доставляется в P_i только в том случае, если содержимое заголовка соответствует этим условиям, иначе блок mail_i ждет другое сооб-

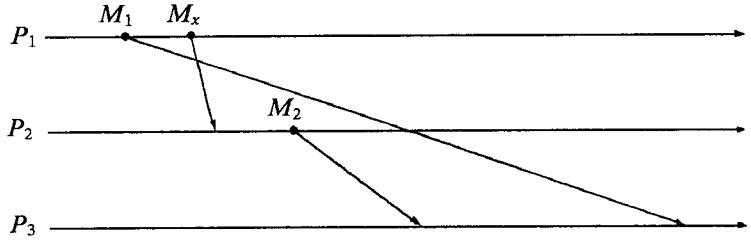


Рис. 1. Пример нарушения причинного порядка событий

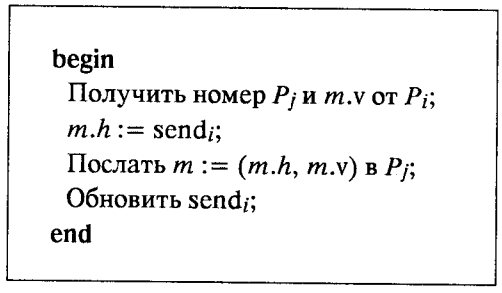


Рис. 2. Посылка сообщения m из mail_i

щение, а m помещается в очередь. Алгоритмы различаются в основном структурой данных заголовка и разрешающих условий.

На рис. 2 представлена схема выполнения блока mail_i при посылке основного значения m.v из процесса P_i в P_j. Блок получает от P_i номер процесса-адресата и m.v.

Локальная переменная send_i, которая может иметь комбинированную структуру, используется для накопления и передачи сведений об известных процессу P_i уже состоявшихся посылках сообщения. Значение этой переменной берется в качестве заголовка $m.h$, и осуществляется посылка сообщения в процесс P_j. Обновление переменной send_i состоит в присоединении к ней сведений об этой посылке.

На рис. 3 представлена схема выполнения блока mail_i при доставке в P_i основного значения m.v принятого сообщения m. Блок mail_i состоит из трех подблоков:

— работа с очередью сообщений;

— ожидание сообщения;

— анализ заголовка сообщения.

Общими переменными подблоков являются: переменная k для хранения текущего числа событий в очереди и булева переменная v для указания результата блока анализа.

Процесс P_i запрашивает прием очередного значения через блок mail_i. Если очередь пуста, то выполняется подблок ожидания сообщения. При получении сообщения управление получает подблок анализа.

Если содержимое заголовка удовлетворяет разрешающим условиям, то:

1) основное значение m.v доставляется в P_i;

2) обновляются локальная переменная send_i и разрешающие условия encon_i в соответствии со сведениями из полученного заголовка $m.h$ сообщения m;

3) булева переменная v принимает значение true, указывая на то, что сообщение доставлено;

4) затем осуществляется выход из mail_i.

В случае несоответствия содержимого заголовка разрешающим условиям выполняется следующее:

1) переменная v принимает значение false, указывая, что сообщение не может быть доставлено в данный момент;

2) сообщение m помещается в очередь;

3) переменная k увеличивается на 1;

4) выполняется подблок ожидания очередного сообщения.

Еще раз отметим, что каналы предполагаются надежными с произвольной, но конечной задержкой связи. Однако возможно и использование подблока, контролирующего интервалы ожидания сообщения и выдающего идентифицирующую информацию при их нарушении.

Если при запросе сообщения процессом P_i очередь не пуста, то анализируются накопленные сообщения либо пока не выберется сообщение, для которого анализ будет успешен, либо пока не исчерпается вся очередь. В первом

```
Общие переменные:
 k — число сообщений в queue_i; v — булева
begin
 if k ≠ 0 then
 begin
  for u := 1 to k do
   begin
    Выбор m_u из queue_i;
    Анализ (m_u);
    if v = true then
     begin
      Удалить m_u; k := k − 1;    Выход;
     end
    fi;
   end
  end
 fi;

1. Ждать сообщения m;
 Анализ (m);
 if v = true    then    Выход;    fi;
 Поместить m в очередь; k := k + 1;
 Переход на 1;
end

  Анализ (m)
  begin
   if m.h соответствует encon_i then
    begin
     Доставить m.v в P_i;
     Обновить send_i и encon_i; v := true;
    end
   else v := false;
   fi;
  end
```

Рис. 3. Прием сообщения m в mail_i и доставка $m.\text{v}$ в P_i

случае сообщение удаляется из очереди, значение переменной k уменьшается на 1 и происходит выход из блока mail_i. Во втором случае управление получает подблок ожидания сообщения.

Как уже было отмечено, алгоритмы в основном различаются структурами локальной переменной send_i и разрешающих условий encon_i. В связи с этим в рамках общей рассмотренной схемы алгоритмы различаются функциями проверки и обновления таких переменных, выполняемыми подблоком анализа.

3. Новый алгоритм сохранения причинного порядка событий. Для описания алгоритма введем следующие обозначения. Посылку сообщения M посредством события e_{ij} обозначим через $S(M_{ij})$, а его доставку в некоторый процесс

```
begin
  Получить номер P_k и m.v от P_i;
  ac_i := ac_i + 1;
  m.h := (ac_i, ORD_BUF_P_i);
  M_ij := (m.h, m.v);
  Послать M_ij в P_k;
  Обновить ORD_BUF_P_i;
end
```

Рис. 4. Посылка в P_k сообщения M_{ij} из mail_i

P_s после установления соответствия содержимого $m.h$ условиям encon_i — через $D(M_{ij})$, полагая, что в P_s доставляется основное значение $m.v$.

Блок mail_i управляет тремя переменными, из которых первые две представляют переменную send_i, а третья — разрешающие условия encon_i:

ac_i — порядковый номер посылки $S(M_{ij})$ (вначале $ac_i = 0$);

ORD_BUF_P_i — упорядоченный буфер элементов (P_k, P_r, ac_r), где P_k, P_r — соответственно номера процесса адресата и источника, включая $r = i$, а ac_r — порядковый номер $S(M_{ry})$ (элементы буфера упорядочиваются по номерам P_k, вначале он пуст);

DELIV_i — массив $[1, ..., N]$ целых чисел (вначале $\forall_g \text{DELIV}_i[g] = 0$).

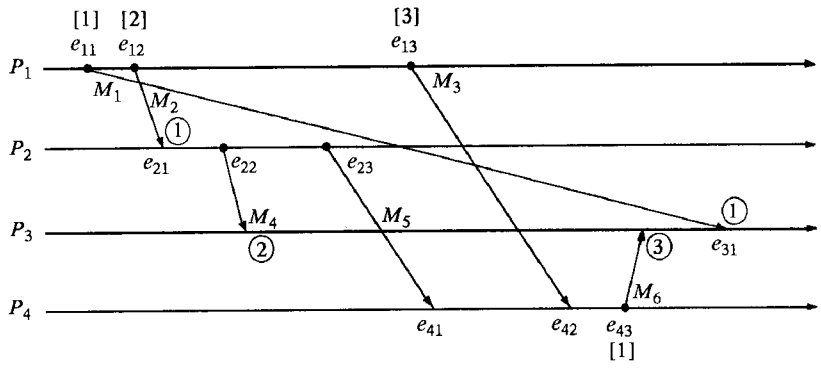
На рис. 4 представлена схема выполнения блока mail_i при посылке сообщения M_{ij} из процесса P_i в P_k.

Обновление ORD_BUF_P_i после посылки $S(M_{ij})$ из P_i в P_k сводится к включению в него нового элемента (P_k, P_i, ac_i). Если буфер уже содержит элементы вида $(P_k, ..., ...)$, то все они заменяются этим элементом, поскольку информация о них уже отправлена процессу P_k вместе с $S(M_{ij})$. Таким образом, информация о посылке $S(M_{ij})$ будет распространяться с очередными сообщениями из P_i.

На рис. 5 представлена схема выполнения блока mail_i при приеме сообщения M_{kj} процессом P_i от процесса P_k. Заголовок сообщения $m.h = (ac_k, \text{ST}M_k)$ содержит номер $S(M_{kj})$ и буфер $\text{ST}M_k$, несущий содержимое

```
Анализ (m)
begin
 if  G = ∅ ∨ ∀ ac_r DELIV_i[r] ≥ ac_r  then
 begin
  D(M_kj);
  DELIV_i[k] := ac_k;
  Обновить ORD_BUF_P_i;
  v := true;
 end
 else  v := false;  fi;
end
```

Рис. 5. Прием сообщения M_{kj} в mail_i и доставка $m.v$ в P_i



ORD_BUF_P_1	
перед S M_1:	пустой
перед S M_2:	$(P_3, P_1, 1)$
перед S M_3:	$(P_2, P_1, 2)$
	$(P_3, P_1, 1)$
после S M_3:	$(P_2, P_1, 2)$
	$(P_3, P_1, 1)$
	$(P_4, P_1, 3)$

ORD_BUF_P_2	
после D M_2:	$(P_3, P_1, 1)$
перед S M_5:	$(P_3, P_2, 1)$
после S M_5:	$(P_3, P_2, 1)$
	$(P_4, P_2, 2)$

ORD_BUF_P_4	
после D M_5:	$(P_3, P_2, 1)$
после D M_3:	$(P_2, P_1, 2)$
	$(P_3, P_1, 1)$
	$(P_3, P_2, 1)$
после S M_6:	$(P_2, P_1, 2)$
	$(P_3, P_4, 1)$

ORD_BUF_P_3	
после D M_1:	пустой
после D M_4:	пустой
после D M_6:	$(P_2, P_1, 2)$

Рис. 6. Пример сохранения причинного порядка событий

ORD_BUF_P_k. Буфер STM_k может содержать некоторое множество G элементов вида (P_i, P_r, ac_r), которые содержат информацию о $S(M_{ry})$ в P_i от процессов $P_r \in \{P_1, P_2, ..., P_N\}$, либо такие элементы отсутствуют. Доставка сообщения в P_i от P_k может быть разрешена, только если либо $G = \varnothing$, либо для всех ac_r, входящих в элементы $(P_i, P_r, ac_r) \in G$, выполняется условие DELIV$_i[r] \geq ac_r$.

Обновление ORD_BUF_P_i после $D(M_{kj})$ в P_i от P_k сводится к его слиянию с элементами (P_a, P_b, ac_b), где $a \neq i$, из STM_k. Такие элементы просто добавляются в ORD_BUF_P_i, если элементы с таким же адресатом P_a или отсутствуют, или их источник отличен от P_b. Если в нем имеется элемент вида (P_a, P_b, ac_{b_c}), то он заменяется на элемент $(P_a, P_b, \max(ac_{b_c}, ac_b))$.

Таким образом, буфер ORD_BUF_P_i в максимальном случае может содержать информацию для $(N - 1)$ процессов P_k, где $k \neq i$, и для каждого из P_k иметь $(N - 1)$ элементов вида (P_k, P_r, ac_r), где $k \neq r$.

4. **Анализ объемов дополнительной информации.** На рис. 6 представлена пространственно-временная диаграмма, описывающая ход выполнения четырех процессов, взаимодействия между которыми представляют несколько

типичных ситуаций и управляются в соответствии с рассмотренным алгоритмом. Цифры в кружках для процесса P_3 показывают реальный порядок приема сообщений этим процессом, который удовлетворяет требованию согласованности с причинным порядком их посылки.

В нижней части рис. 6 показано содержимое упорядоченных буферов ORD_BUF_P_i ($i = 1, 2, 3, 4$) перед посылкой (перед S), после посылки (после S) и после доставки (после D) некоторых сообщений. Например, буфер процесса P_1 перед посылкой сообщения M_1 пуст, а перед посылкой M_2 он содержит элемент, указывающий, что процесс P_1 выполнил посылку сообщения с порядковым номером $ac_1 = 1$ в процесс P_3. Перед посылкой сообщения M_3 буфер содержит уже два элемента, упорядоченных по номерам процессов-адресатов.

Данный пример взят из работы [2] для иллюстрации нового содержимого дополнительной информации. Проведем сравнение ее объемов $Q1_i$ и $Q2_i$, используемых соответственно в новом алгоритме и алгоритме из [2] при посылке сообщения из P_i в процесс P_x.

Напомним, что в предложенном алгоритме $m.h = (ac_i, \mathrm{ST}M_i)$, где ac_i — порядковый номер посылки из P_i, а $\mathrm{ST}M_i$ содержит элементы (P_k, P_r, ac_r) буфера ORD_BUF_P_i на момент посылки, который в максимальном случае может содержать информацию для $(N - 1)$ процессов-адресатов P_k, где $k \neq i$, и для каждого из P_k иметь $(N - 1)$ процессов-источников, где $r \neq k$.

Обозначим число процессов-адресатов, о которых известно процессу P_i, через f $(0 \leq f \leq (N - 1))$, а число процессов-источников для каждого из таких процессов через z_c $(1 \leq z_c \leq (N - 1);\ 1 \leq c \leq f)$, тогда $Q1_i = 1 + 3\sum_{c=1}^{f} z_c$ или $Q1_i = 1 + 3zf$, если за z примем некоторое среднее число процессов-источников для каждого из f процессов. При $z = N/3$ $Q1_1 = 1 + Nf$.

В алгоритме из [2] $m.h = (V_i, \mathrm{ST}M_i)$, где V_i — логическое время в момент посылки из P_i, представляемое одномерным массивом размерностью N целых чисел, а $\mathrm{ST}M_i$ содержит элементы (P_k, V_j), где P_k есть также номер процесса-адресата, а V_j — логическое время, несущее информацию о всех процессах-источниках для P_k, известных в момент посылки из процесса P_j. Объем $Q2_i = N + (1 + N)f$ независимо от числа z процессов-источников, поэтому, если число таких процессов меньше $(N - 1)$, информация будет избыточной. При $z = N/3$ и при одних и тех же значениях f объем $Q2_i$ будет больше $Q1_i$ не менее чем на N. Кроме того, эта информация вместе со всем сообщением m может занимать пространство в очереди. Поэтому для задач, у которых $z \leq N/3$, новый алгоритм более приемлем.

Алгоритм из [3] более прост для понимания и реализации, однако заголовок $m.h$ каждого сообщения содержит матрицу $[N \times N]$ целых чисел. Алгоритм же из [2] достигает таких размеров только при $f = (N - 1)$.

Следует также отметить, что предложенный алгоритм ориентирован на управление причинным порядком событий именно в процессе счета распределенной программы. Поэтому в целях снижения размерности пересылаемых чисел определяются порядковые номера событий только типа посылки сообщения. При использовании алгоритма и при отладке такой программы может потребоваться идентификация событий любого из трех типов.

5. **Доказательство корректности.** Докажем корректность алгоритма на основе доказательства его безопасности и живучести. Безопасность соответствует сохранению причинного порядка событий при управлении взаимодействиями процессов данным алгоритмом, живучесть — тому, что каждое $m.v$ будет доставлено.

Вначале докажем несколько необходимых далее свойств вспомогательной информации, следующих из алгоритма и отношения «случилось перед».

Свойство 1. Если $S(M_{kx})$ есть посылка в P_i из процесса P_k такая, что буфер ORD_BUF_P_k, посылаемый в заголовке $m.h$ вместе с сообщением M_{kx}, содержит элемент (P_i, P_j, ac_j) (включая $j = k$), то $S(M_{jz}) \rightarrow S(M_{kx})$.

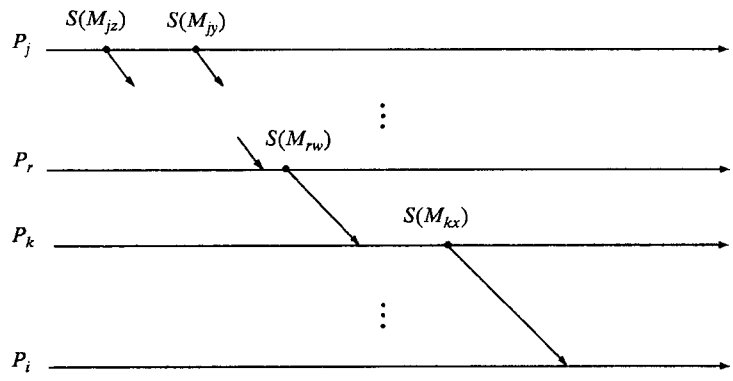


Рис. 7. Упорядоченная цепочка событий посылки сообщений $S(M_{jz}) \rightarrow \ldots \rightarrow S(M_{rw}) \rightarrow S(M_{kx})$

Доказательство. (*i*) $j = k$. По алгоритму переменная ac_j всегда увеличивается на 1 при посылке сообщения из процесса P_j и ее значение присоединяется к содержимому буфера вместе с элементом (P_i, P_j, ac_j) только после посылки. Следовательно, когда происходит $S(M_{kx})$, процесс P_k уже выполнил ac_j-ю посылку, т. е. $S(M_{jz}) \rightarrow S(M_{kx})$.

(*ii*) $j \neq k$. Доказательство будет состоять в построении упорядоченной цепочки $S(M_{jz}) \rightarrow \ldots \rightarrow S(M_{rw}) \rightarrow S(M_{kx})$ (рис. 7). Так как буфер ORD_BUF_P_k уже содержит элемент (P_i, P_j, ac_j) и $j \neq k$, следовательно, такой элемент был доставлен в заголовке сообщения от некоторого другого процесса (возможно, не одного). Пусть первое из таких сообщений поступило от процесса P_r при посылке $S(M_{rw})$. Следовательно, перед $S(M_{rw})$ процесс P_r уже имел этот элемент в своем буфере ORD_BUF_P_r. Далее можно провести аналогичные рассуждения относительно процесса P_r. Так как число процессов конечное, а отношение «случилось перед» определяет порядок на событиях, то построение такой цепочки завершится выбором некоторого события посылки $S(M_{jy})$, которую уже можно трактовать посредством (*i*). □

Свойство 2. Если $S(M_{jz})$ есть посылка в P_i, то любое другое сообщение M_{kx} такое, что $S(M_{jz}) \rightarrow S(M_{kx})$ (включая $k = j$), будет содержать в своем заголовке $m.h$ или элемент (P_i, P_j, ac_j), или элемент (P_i, P_r, ac_r), если $S(M_{rd})$ есть посылка в P_i и $S(M_{jz}) \rightarrow S(M_{rd}) \rightarrow S(M_{kx})$.

Доказательство. (*i*) $k = j$. Из алгоритма следует, что после каждой очередной посылки из P_j в один и тот же процесс P_i элемент (P_i, P_j, ac_j) в ORD_BUF_P_j заменяется новым с увеличенным значением ac_j.

(*ii*) $k \neq j$. Если в упорядоченной цепочке событий $S(M_{jz}) \rightarrow \ldots \rightarrow S(M_{rw}) \rightarrow \rightarrow S(M_{kx})$, связывающей событие $S(M_{jz})$ с событием $S(M_{kx})$, некоторый процесс P_r выполнил в P_i посылку $S(M_{rd})$ такую, что $S(M_{jz}) \rightarrow \ldots \rightarrow S(M_{rd}) \rightarrow S(M_{rw})$, то по алгоритму перед событием $S(M_{rw})$ буфер ORD_BUF_P_r будет содержать элемент (P_i, P_r, ac_r) вместо (P_i, P_j, ac_j). □

Свойство 3. Если $S(M_{jz})$ есть посылка из процесса P_j в процесс P_i, а ac_j — текущий номер этой посылки из P_j, то $\mathrm{DELIV}_i[j] < ac_j$ до тех пор, пока не будет выполнена доставка $D(M_{jz})$ в P_i.

Доказательство. Рассмотрим только сообщения, посылаемые из P_j в P_i, поскольку лишь доставка таких сообщений может изменить значение компоненты $\mathrm{DELIV}_i[j]$. Каждое из этих сообщений содержит заголовок $m.h = (ac_j, \mathrm{ST}M_j)$, где ac_j есть порядковый номер текущей посылки сообщения из P_j, который становится новым значением компоненты $\mathrm{DELIV}_i[j]$ только после доставки этого сообщения. Поэтому единственным путем иметь $\mathrm{DELIV}_i[j] \geq ac_j$ является доставка $D(M_{jz})$. □

Безопасность. Покажем по индукции на событиях процесса адресата P_i, что если $S(M_{jz})$ есть посылка сообщения в P_i, то ни одно сообщение M_{kx} в P_i

такое, что $S(M_{jz}) \rightarrow S(M_{kx})$, не будет доставлено в P_i до тех пор, пока не будет доставлено в P_i сообщение M_{jz}.

1. *Основной шаг.* Предположим, что одно из M_{kx} сообщений является самым первым из доставленных в P_i. Перед его доставкой переменная $\mathrm{DELIV}_i = 0$, так как она обновляется только после доставки. Из условия $S(M_{jz}) \rightarrow S(M_{kx})$ и свойства 2 следует, что заголовок $m.h$ сообщения M_{kx} содержит или элемент (P_i, P_j, ac_j), или (P_i, P_r, ac_r). Так как все компоненты переменной DELIV_i равны нулю, то условие доставки не выполнено. Значит, сообщение M_{kx} не может быть доставлено, что противоречит нашему предположению.

2. *Шаг индукции.* Предположим в качестве гипотезы индукции, что ни одно из $(n-1)$ уже доставленных сообщений в P_i не является таким M_{rd} сообщением, что $S(M_{jz}) \rightarrow S(M_{rd})$. Покажем, что если M_{jz} еще не доставлено в P_i, то $D(M_{kx})$ не может быть и n-м событием доставки в P_i. По тем же причинам, что и в основном шаге, заголовок $m.h$ сообщения M_{kx} содержит или элемент (P_i, P_j, ac_j), или (P_i, P_r, ac_r). Так как M_{jk} еще не доставлено, то из свойств 2, 3 и гипотезы индукции следует, что $\mathrm{DELIV}_i[j] < ac_j$ или $\mathrm{DELIV}_i[r] < ac_r$. Следовательно, и в этом случае условие доставки не выполнено, значит, $D(M_{kx})$ не может быть n-м событием доставки в P_i.□

Живучесть. Доказательство в основном совпадает с [2, 3]. Исходя из предположения, что управляемая алгоритмом программа выполняется без аварийных ситуаций с использованием надежных каналов с произвольной, но конечной задержкой связи, можно сделать вывод, что все посланные любому из процессов сообщения будут приняты. Допустим, что некоторые из таких сообщений не доставлены и сохранены в очереди. Среди них найдется самое раннее в смысле отношения «случилось перед». Если условия доставки для него не удовлетворяются, то по свойству 1 существует еще более раннее сообщение в указанном смысле. Однако это противоречит выводу о всех принятых сообщениях. Следовательно, такое сообщение может быть доставлено, что повлечет за собой и доставку остальных сообщений из очереди.□

```
Анализ (m)
begin
 if cur.iter = num.iteration then
 begin
  Доставка (index.x, m.v) в Pi;
  sum.p := sum.p + 1;
  if  sum.p = n  then
   begin
    sum.p := 0;
    cur.iter := cur.iter + 1;
   end
  v := true;
 end
 else  v := false;  fi;
end
```

Рис. 8. Доставка $x_j^{(k)}$ в P_i

6. Пример: решение системы n линейных уравнений методом релаксации Якоби. i-е уравнение представляется $\sum_{j=1}^{n} a_{ij}x_j = b_i$. $(k+1)$-е итеративное значение x_i есть

$$x_i^{(k+1)} = \frac{1}{a_{ij}} \left(b_i - \sum_{j=1,\, j \neq i}^{n} a_{ij}\, x_j^{(k)} \right).$$

Начальное значение каждого x_i равно 0. i-й процесс вычисляет $x_i^{(k)}$ и хранит a_{ij}, $1 \le j \le n$. Вначале i-й процесс вычисляет $x_i^{(1)}$ при $x_j^{(0)} = 0$, $1 \le j \le n$, $i \ne j$.

Чтобы вычислить $x_i^{(k+1)}$, i-й процесс должен принять значения $x_j^{(k)}$, $1 \le j \le n$, $i \ne j$ от других процессов. Однако может возникнуть ситуация, когда значения $(k+1)$-й итерации некоторых процессов могут быть приняты прежде (k)-й, если не учитывать возможные нарушения порядка приема сообщений.

На рис. 8 приводится подблок анализа, который в совокупности с остальными подблоками блока mail_i гарантирует, что все доставленные x_j, $1 \le j \le n$, $i \ne j$ принадлежат одной и той же итерации. Структура заголовка сообщения: $m.h$ = (index.x, num.iteration), где первая переменная равна j, а вторая — номеру текущей итерации в процессе P_j. Отметим, что порядок приема x_j, соответствующих одной и той же итерации, может быть произвольным, так как выбор a_{ij} базируется на значении переменной index.x. Роль управляющих условий выполняют две переменные:

sum.p — число процессов, исходное значение: sup.p = 0;

cur.iter — текущая итерация, исходное значение: cur.iter = 1.

Данный пример показывает, что объем дополнительной информации может содержать всего два целых числа, если учитывать свойства конкретной задачи.

Заключение. Практическое использование исследованных алгоритмов зависит в основном от объема информации в заголовке $m.h$, который, в свою очередь, тесно связан с числом z процессов-источников предшествующих посылок, о которых необходимо сообщить адресату. Алгоритмы из [1—3] ориентированы на сохранение потенциально возможных причинных порядков событий, поэтому объем вспомогательных данных может быть избыточным. Предложенный алгоритм ориентирован на сохранение реально возникающих в ходе выполнения программы причинных порядков событий, поэтому, как было показано, он может быть более приемлем для задач, у которых $z \le N/3$. Кроме того, пример из предыдущего раздела демонстрирует, что интеграция общей схемы таких алгоритмов со свойствами конкретной задачи может существенно снизить объемы вспомогательных данных.

СПИСОК ЛИТЕРАТУРЫ

1. **Birman K., Joseph T.** Reliable communications in the presence of failures // ACM Trans. on Comput. Syst. 1987. **5**, N 1. P. 47.
2. **Schiper A., Eggli J., Sandoz A.** A new algorithm to implement causal ordering // Lecture Notes in Computer Science. 1989. N 392. P. 219.
3. **Raynal M., Schiper A., Toueg S.** The causal ordering abstraction and a simple way to implement it // Inf. Proc. Lett. 1991. **39**, N 6. P. 343.
4. **Lamport L.** Time, clocks and the ordering of events in a distributed system // Commun. ACM. 1978. **21**, N 7. P. 558.

Поступила в редакцию 22 января 1996 г.