**Вариант 7**

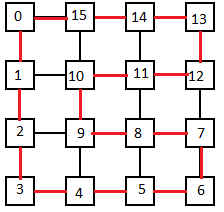
1. **Сколько времени потребует выбор координатора среди 16 процессов, находящихся в узлах транспьютерной матрицы размером 4\*4, если используется круговой алгоритм? Время старта равно 200, время передачи байта равно 1 (Ts=200,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:

*Круговой алгоритм.*

Алгоритм основан на использовании кольца (физического или логического). Каждый процесс знает следующего за ним в круговом списке. Когда процесс обнаруживает отсутствие координатора, он посылает следующему за ним процессу сообщение «ВЫБОРЫ» со своим номером. Если следующий процесс не отвечает, то сообщение посылается процессу, следующему за ним, и т.д., пока не найдется работающий процесс. Каждый работающий процесс добавляет в список работающих свой номер и переправляет сообщение дальше по кругу. Когда процесс обнаружит в списке свой собственный номер (круг пройден), он меняет тип сообщения на «КООРДИНАТОР» и оно проходит по кругу, извещая всех о списке работающих и координаторе (процессе с наибольшим номером в списке). После прохождения круга сообщение удаляется.

Пусть кольцо имеет следующий вид. Пусть сообщение «ВЫБОРЫ» занимает 1 байт. Под номер процесса тоже отведем 1 байт для простоты.



Процесс 0 отправляет процессу 1 «ВЫБОРЫ» + свой номер. Процесс добавляет к полученному сообщению свой номер, и отправляет процессу 2 и т.д. Для полного прохождения круга надо отправить 16 сообщений. Размер сообщения с каждым разом будет расти на 1 байт, и составит арифметическую прогрессию 2, 3, …, 17. Один круг «ВЫБОРОВ» займет T1 = (Ts+2\*Tb)+(Ts+3\*Tb)+…+(Ts+17\*Tb)=16\*Ts+152\*Tb. После этого процесс 0 найдет себя в списке, обнаружит что круг пройден, выберет максимальный номер среди номеров процессов (в данном случае 15), сформулирует сообщение «КООРДИНАТОР» + номер координатора, и отправит процессу 1. Процесс 1 в свою очередь отправляет процессу 2, и т.д. Под «КООРДИНАТОР» отведем 1 байт для простоты. Тогда размер соощения будет 2 байта. Для полного прохождения круга потребуется 16 сообщений. Это займет T2=16\*(Ts+2\*Tb) = 16\*Ts+32\*Tb.

Ответ: Общее время Т = Т1+Т2 = 32\*Ts+184\*Tb = 32\*200+184=6584.

**===отлично**

1. **Консистентность памяти по входу и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует трехкратное выполнение критической секции и модификация в ней 11 переменных каждым процессом, если DSM реализована на 10 ЭВМ сети с шинной организацией (с аппаратными возможностями широковещания). Время старта (время «разгона» после получения доступа к шине для передачи) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи операции посылки сообщения (при одновременно выданных операциях - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:  
Вводится новый вид переменной - синхронизационная переменная, связанная с общими переменными.

Каждая переменная синхронизации имеет текущего владельца — процесс, который захватил ее последним. Владелец может многократно входить в критические области и выходить из них, не посылая в сеть никаких сообщений. Процесс, не являющийся в настоящее время владельцем переменной синхронизации, но желающий захватить ее, должен послать текущему владельцу сообщение, запрашивая право собственности и текущие значения данных, ассоциированных с переменной синхронизации.

Критические секции, охраняемые одной синхронизационной переменной, могут быть двух типов:

1. секция с монопольным доступом (для модификации переменных);
2. секция с немонопольным доступом (для чтения переменных).

Разрешена ситуация, когда синхронизационная переменная имеет несколько владельцев, если связанные с этой переменной общие данные используются только для чтения.

Алгоритм (децентрализованный). У каждого процесса есть своя очередь запросов на доступ к синхронизационной переменной. Запросы бывает двух типов: запрос на монопольный доступ и немонопольный доступ. Для получения права владения синхронизационной переменной, процесс выполняет неделимый широковещательный запрос с указанием типа запроса. При получении запроса, процесс добавляет запрос в очередь. Если процесс не выдавал запрос, то он отвечает на все запросы "ОК". Если процесс выдавал запрос на монопольный доступ, то отвечает всем запросам, стоящим перед ним в очереди "ОК" (себе тоже для удаления из очереди). Если процесс выдавал запрос на немонопольный доступ, то отвечает всем запросам, стоящим перед ним в очереди "ОК", а также всем стоящим после него немонопольным запросам до первого запроса в монопольным режиме. Процесс в обоих случаях отправляет "ОК" себе тоже. При отправке "ОК" какому-то запросу в очереди, этот запрос удаляется из очереди. Процесс получает право собственности на переменную в соответствующем режиме при получении «ОК» от всех процессов. При выходе из КС, процесс отправляет «ОК» всем остальным процессам в очереди. При выходе из КС, процесс, владеющий монопольным правом собственности на переменную, помимо «ОК», отправляет текущие значения переменных связанных с синхронизационной переменной, всем немонопольным запросам в очереди до первого монопольного запроса включительно. Т.е. каждый процесс имеет очередь запросов, и она продвигается аналогично очереди координатора в предыдущем алгоритме, но у каждого процесса независимо. Порядок запросов будет совпадать у всех процессов в силу неделимых широковещательных рассылок - это не так, надо было использовать неделимые широковещательные рассылки (рассчитывать на особенности коммуникаций в алгоритме нельзя – эти особенности надо учитывать только при вычислении времен!) . Да, надо использовать неделимую широковещательную рассылку.

Для разных синхронизационных переменных заводятся разные очереди.

1) Консистентность по входу - совместно используемые данные, относящиеся к данной критической области, становятся непротиворечивыми при входе в эту область.

2)

а) пишем в локальную память;

б) читаем из локальной памяти;

в) значения модифицируемых переменных рассылается всем немонопольным запросам в очереди до первого монопольного запроса включительно, процессом владеющим переменным в монопольном режиме при выходе из КС; **(Разослали всем в очереди, потом появился новый читатель – ему уже не пошлют актуальные данные!)** К алгоритму надо добавить следующий пункт: при получении любого «ЗАПРОСА», если мы предыдущий монопольный владелец синхронизационной переменной, и если еще не отправляли актуальные значения какому-либо монопольному (А немонопольному?) запросу, то помимо «ОК» надо передать и актуальные значения.

г) процесс блокируется пока не получит актуальные значения переменных связанных с синхр. переменной от предыдущего монопольного владельца, и «ОК» от всех процессов.

д) при входе в КС широковещательно рассылается запрос на право владения синхр. переменной в соответствующем режиме (монопольный/ немонопольный). При выходе из КС отправляется «ОК» всем процессам в очереди. Процесс, владеющий синхр. переменной в монопольном режиме, также рассылает текущие значения переменных, связанных с синхр переменной до первого запроса в монопольном режиме включительно.

3. Оценка времени:

В этой задаче все запросы на монопольный доступ. Оценим время одного прохождения КС. Надо сделать одну широковещательную рассылку. Принять 8 «ОК» **ПОЧЕМУ 8?**. 8, т.к. всего 10 ЭВМ, 1 ЭВМ мы сами, осталось 9 ЭВМ. 8 из них будет просто «ОК», одно сообщение «ОК», еще плюс актуальные значения. Одно сообщение «ОК» + актуальные значения связанных переменных. Всего прохождений КС = 3\*10=30. Пусть под «ОК» и тип запроса отведем по 1 байту, адрес и значение переменных тоже по 1 байту. В запросе указывается: тип запроса, и синхр. переменная (адрес и значение).

Запрос = тип запроса+адрес и значение синхр. переменной = 3 байта.

ОК = 1байт.

ОК+ актуальные значения = 1байт + 12 (**ПОЧЕМУ 12?** 12, т.к. 11 модифицируемых переменных + 1 синхронизационная переменная**)** \* 2байта =25 байтов.

Итого Т = Ts + 3b\*Tb + 8\*(Ts+1b\*Tb) + Ts + 25b\*Tb = 100+3+8\*(100+1)+100+25 = 1036.

Пусть в начале у процесса 0 находится монопольное право на синхр. переменную. Тогда ему не придется делать запрос и ждать ответов. **ПОЧЕМУ – ведь трехкратное выполнение критической секции и модификация в ней 11 переменных каждым процессом. Если бы Вы выбрали маркерный алгоритм, тогда бы для одного процесса вход в критическую секцию не потребовал бы сообщений.**

Точно, по приведенному выше алгоритму всегда делаем неделимый широковещательный запрос перед входом в КС.

1. Если предыдущий монопольный владелец синхронизационной переменной сам процесс, то не будем перессылок актуальных значений, т.к. уже имеем. Поэтому надо получить просто 9 «ОК».
2. А если предыдущий владелец другой процесс, тогда перессылок актуальных значений не избежать.

В зависимости от этого, общее время будет Т = Ts + 3b\*Tb + 9\*(Ts+1b\*Tb) + 29\*1036 = 30044 + 1012 = 31056 в первом случае, и Т = 1036\*30 = 31080 во втором случае.

Тогда общее время 29\*1036 = 30044.  
**===хорошо**

Я не могу считать алгоритм правильным, есть вопрос об определении предыдущего владельца – ведь о входе в КС и выходе из нее процессы другим не сообщают. Но идея заменить логическое время на неделимые рассылки – она интересная.

Гораздо более простое решение – взять за основу маркерный алгоритм и передавать вместе с маркером требуемую информацию – очередь на вход в монопольную и немонопольную КС и количество читателей.

1. **Консистентное и строго консистентное множества контрольных точек. Дайте оценку накладных расходов на синхронную фиксацию строго консистентного множества контрольных точек для сети из 10 ЭВМ с шинной организацией (без аппаратных возможностей широковещания), если накладные расходы на синхронную фиксацию консистентного множества равны Т1. Время старта (время «разгона» после получения доступа к шине для передачи) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи операции посылки сообщения (при одновременно выданных операциях - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:

Сначала выполняем алгоритм фиксации консистентного множества контрольных точек. Эти контрольные точки будем считать промежуточными. Исходя из определения, для того, чтобы консистентное множество точек стало строго консистентным, надо убедиться, что между процессами нет никаких сообщений. Для этого мы можем просто пропустить по всем каналам свои собственные сообщения. Если они все пройдут, значит,

каналы пусты и множество строго консистентно. Однако, стоит обратить внимание, что координатор уже посылал всем служебные сообщения, так что его каналы проверять не нужно. У нас остается 9 ЭВМ, которые хотят проверить по 8 каналов каждая. ЭВМ запоминают, по каким каналам им приходят эти служебные сообщения. Если придут по всем 8, посылают сообщение координатору с указанием того, что они

готовы к созданию точки. Если координатору придут все сообщения, он рассылает уведомление о фиксации множества контрольных точек.

SM – ServiceMessage;

IM – ImplementMessage.

T1 – фиксация консистентного множества контрольных точек;  
9\*8\*(Ts+Tb\*SIZESM) – проверка каналов;

9\*(Ts+Tb\*SIZEOK) – уведомления о готовности;

9\*(Ts\*Tb\*SIZEIM) – фиксация пробных точек.

Пусть все служебные сообщения занимают по 1 байту.

Ответ: T = T1 + 10\*(Ts+Tb) = T1 + 1010

Решение грубое, прошу дать мне подсказку для движения в верном направлении, ибо я лучше не придумал. **Надо каждому процессу сообщить, сколько ему было направлено сообщений от других процессов – пусть ждет их все.**

**Алгоритм.**

1-ая фаза

Инициатор сообщает всем о начале создания процесса контрольной точки. После получения сообщения о начале создании контрольной точки, процессу запрещается посылать неслужебные сообщения (инициатору тоже запрещается). Каждый процесс отвечает инициатору списком номеров процессов и сколько сообщений этим процессам было отправлено. Инициатор получает от всех ответы, после этого формирует для каждого процесса сообщение из списка номеров процессов, и сколько сообщений этими процессами было отправлено данному процессу. После принятия всех сообщений, процесс (в том числе инициатор) создает пробную контрольную точку и сообщает об успехе (или о неуспехе, если процесс не смог создать пробную контрольную точку) инициатору. Если все процессы создали контрольные точки, то инициатор принимает решение о превращении пробных контрольных точек в постоянные. Если какой-либо процесс не смог сделать пробную точку, то принимается решение об отмене всех пробных точек.

2-ая фаза

Инициатор информирует все процессы о своем решении. В результате либо все процессы будут иметь новые постоянные контрольные точки, либо ни один из процессов не создаст новой постоянной контрольной точки. Только после выполнения принятого процессом инициатором решения все процессы могут посылать сообщения.

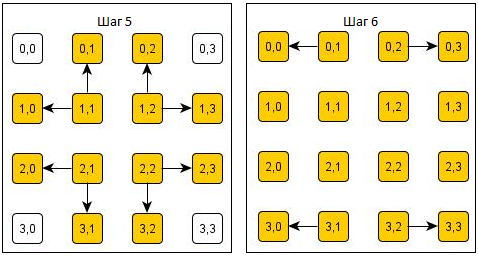
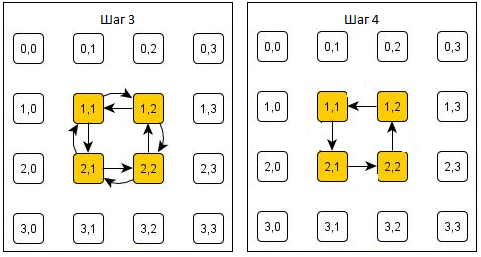
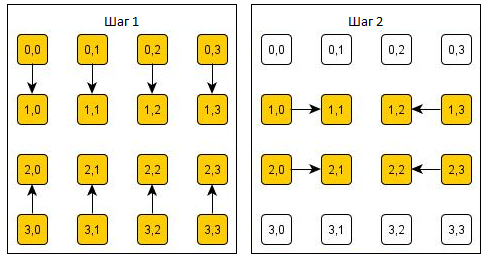
Данный алгоритм отличается от алгоритма для создания просто консистентного, что надо дополнительно отправлять списки процессов и количества сообщений. Пусть под номер процесса и количество сообщений отведем по 1 байту. Тогда сообщения со списком процессов и количеством сообщений от данного процесса занимают 9\*(1+1)= 18 байтов. 9 – потому что всех процессов – 10, и надо знать сколько сообщений ожидать от всех процессов кроме себя. А инициатору тоже отправляется список кому и сколько сообщения было отправлено, тоже кроме себя. Такие обмены списками происходит дважды между каждым процессом и инициатором. Каждый обмен занимает Т2 = Ts + 18\*Tb. Всего обменов 9\*2 = 18.

Ответ: Общее время Т = Т1 + 18\*Т2 = Т1+ 2124.

**===отлично**

1. **В транспьютерной матрице размером 4\*4, в каждом узле которой находится один процесс, необходимо выполнить операцию нахождения максимума среди 16 чисел (каждый процесс имеет свое число). Сколько времени потребуется для получения всеми максимального числа, если все процессы выдали эту операцию редукции одновременно. А сколько времени потребуется для нахождения максимума среди 64 чисел в матрице 8\*8? Время старта равно 50, время передачи байта равно 1 (Ts=50,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение: Покажем на картинке. Стрелками показаны передачи максимумов.



Как видно, нахождение максимума выполняется за 6 шагов. Шаги 1-4 само нахождение максимума, шаги 5-6 - рассылка. Пусть под число отводится байт.

Ответ: T = 6\*(Ts+1\*Tb) = 6\*(50+1\*1) = 306

В случае 8\*8, модифицируем приведенное решение. Делим на 4 четверти по 4\*4. В каждой четверти выполняем шаги 1-4. Потребуется еще 2 шага для пересылки максимумов каждой четверти к центральным узлам

(от (2,2) => (3,3) и т.д.). Еще 2 шага для нахождения глобального максимума (аналогично шагам 3-4 в случае 4\*4), еще 6 шагов, чтобы разослать результат остальным узлам. Получили 4+2+2+6 = 14 шагов.

Ответ: T = 14\*(Ts+1\*Tb) = 14\*(50+1\*1) = 714

**===отлично**

1. **Последовательная консистентность памяти и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует модификация 10 различных переменных 10-ю процессами (каждый процесс модифицирует одну переменную), находящимися на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания) и одновременно выдавшими запрос на модификацию. Время старта (время «разгона» после получения доступа к шине) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:  
Последовательная консистентность – все процессы видят одну и ту же последовательность записей в память(как будто память едина и в каждый момент времени исполняется ровно одна команда какого-то процесса).

Последовательная консистентность может быть реализована также следующим образом. Страницы, доступные на запись, размножаются, но операции с разделяемой памятью не должны начинаться до тех пор, пока не завершится выполнение предыдущей операции записи, выданной каким-либо процессором, т.е. будут скорректированы все копии соответствующей страницы (все записи выполняются последовательно, блокируя на время своего выполнения работу всех процессов). В системах с упорядоченным механизмом широковещания запрос на операцию модификации памяти рассылается всем владельцам копий соответствующей страницы (включая и себя). При этом, работа процессов не блокируется. Одним процессором могут быть выданы несколько запросов на модификацию данных. Любая операция чтения не должна выполняться до того, как будут выполнены все выданные данным процессором запросы на модификацию (процессор получит и выполнит «свои» запросы).

Централизованный алгоритм. Процесс посылает координатору запрос на модификацию переменной и ждет от него указания о проведении этой модификации. Такое указание координатор рассылает сразу всем владельцам копий этой переменной. Каждый процесс выполняет эти указания по мере их получения. Поскольку сообщения от координатора приходят каждому процессу в том порядке, в котором они были им посланы, то все процессы корректируют свои копии переменных в этом едином порядке.

Реализация последовательной консистентности в DSM с полным размножением:

а) при записи:

* + Посылается координатору запрос на модификацию.
  + Координатор принимает запрос на модификацию, присваивает ей номер и высылает автору номер. Если координатор заведомо знает, что процесс не имеет каких-то модификаций, **ОТКУДА ОН знает – он их не посылал?** от высылает их сразу в этом ответе.
  + Приняв номер, автор должен получить недостающие модификации (если они были потеряны в каналах), затем записать и послать изменение координатору.
  + Координатор рассылает всем (кроме автора) модификацию.
  + Приняв модификацию, процесс проверяет, получил ли он предыдущие модификации, если не получил, то запрашивает их. Применяет все полученные модификации.

б)при чтении:

Производится обращение к локальной копии данных.   
в)значения модифицированных переменных рассылаются координатору после записи(см п (а))

а также координатором автору новых модификаций после получения от него запроса на запись.

г) блокируется ли процесс на время выполнения записи или рассылки значений переменных:

процесс – автор блокируется до записи **(ДО ЧЕГО?)** для получения консистентной версии памяти.

Почему процесс сначала посылает координатору запрос? **Почему сразу не посылает модификацию?**

**Нужен децентрализованный алгоритм прежде всего!**

Всего 10 процессов. Один из них координатор.

Каждый процесс имеет все 10 переменных, всего процессов - 10. Тогда каждый цикл модификации

будет выглядеть как:

· Процесс i посылает координирующему процессу запрос (1 передача запроса размера A)

· Координирующий процесс через какое-то время высылает ответ с подтверждением и порядковым номером

модификации ( 9 передач подтверждения модификации каждой размера B)

Всего у нас будет 9 таких запросов, а поскольку используется шина и все эти соединения должны в любом случае

состоятся, то получаем общее время как сумму времен каждой передачи: **9 \* ( 10 \* Ts + (A + 9 \* B) \* Tb )**

Тогда будем использовать децентрализованный алгоритм (из лекции).   
**Децентрализованный алгоритм.** Процесс посылает посредством механизма упорядоченного широковещания (неделимые широковещательные рассылки) указание о модификации переменной всем владельцам копий соответствующей страницы (включая и себя) и ждет получения этого своего собственного указания.

Т.е. для одной модификации надо сделать одну неделимую широковещательную рассылку.

Так как у нас нет упорядоченного широковещания, придется реализовать это вручную (возьмем алгоритм из лекции).

**Алгоритм надежных неделимых широковещательных рассылок сообщений.**

Алгоритм выполняется в две фазы и предполагает наличие в каждом процессоре очередей для запоминания поступающих сообщений. В качестве уникального идентификатора сообщения используется его начальный приоритет - логическое время отправления, значение которого на разных процессорах различно.

1-ая фаза.

Процесс-отправитель посылает сообщение группе процессов (список их идентификаторов содержится в сообщении).

При получении этого сообщения процессы:

1. Приписывают сообщению приоритет, помечают сообщение как «недоставленное» и буферизуют его. В качестве приоритета используется временная метка (текущее логическое время).
2. Информируют отправителя о приписанном сообщению приоритете.

2-ая фаза.

При получении ответов от всех адресатов, отправитель:

1. Выбирает из всех приписанных сообщению приоритетов максимальный и устанавливает его в качестве окончательного приоритета сообщения.
2. Рассылает всем адресатам этот приоритет.

Получив окончательный приоритет, получатель:

1. Приписывает сообщению этот приоритет.
2. Помечает сообщение как «доставленное».
3. Упорядочивает все буферизованные сообщения по возрастанию их приписанных приоритетов.
4. Если первое сообщение в очереди отмечено как «доставленное», то оно будет обрабатываться как окончательно полученное.

Если получатель обнаружит, что он имеет сообщение с пометкой «недоставленное», отправитель которого сломался, то он для завершения выполнения протокола осуществляет следующие два шага в качестве координатора.

1. Опрашивает всех получателей о статусе этого сообщения.

Получатель может ответить одним из трех способов:

1. Сообщение отмечено как «недоставленное» и ему приписан такой-то приоритет.
2. Сообщение отмечено как «доставленное» и имеет такой-то окончательный приоритет.
3. Он не получал это сообщение.

2. Получив все ответы координатор выполняет следующие действия:

1. Если сообщение у какого-то получателя помечено как «доставленное», то его окончательный приоритет рассылается всем. (Получив это сообщение каждый процесс выполняет шаги фазы 2).
2. Иначе координатор заново начинает весь протокол с фазы 1. (Повторная посылка сообщения с одинаковым приоритетом не должна вызывать коллизий).

Необходимо заметить, что алгоритм требует хранения начального и окончательного приоритетов даже для принятых и уже обработанных сообщений.

Ответ по плану.

1. Результат выполнения должен быть тот же, как если бы операторы всех процессоров выполнялись бы в некоторой последовательности, в которой операторы каждого индивидуального процессора расположены в порядке, определяемом программой этого процессора.
2. а) при записи делается неделимая широковещательная рассылка;

б) читается из локальной памяти;

в) значение модифицируемой переменной рассылается всем неделимо широковещательно при записи;

г) блокируется, т.к. надо ждать завершения неделимой широковещательной рассылки;

1. Оценка времени.

Оценим время одной широковещательной рассылки.

*1 фаза.* Процессов у нас 10. Т.е. текущему процессу надо отправить девятерым остальным процессам адрес и значение модифицируемой переменной, список идентификаторов сообщений кому еще отправили (идентификаторов тоже девять штук). Пусть под адрес, значение и идентификатор отведем по 1 байту. Одно сообщение занимает 11 байтов (если надежность гарантируется, можно список идентификаторов не отправлять и на этом сэкономить. Тогда размер сообщения будет 2 байта.). Отправка одного сообщения займет T1 = Ts+11b\*Tb = 111 (T1 = Ts + 2b\*Tb = 102). Отправка девяти сообщений Т2 = 9\*Т1 = 999 (Т2 = 9\*Т1= 918). Каждый из девяти процессов должен ответить отправителю приписанным приоритетом. Пусть под приоритет отведем 1 байт. Тогда все ответы займут Т3 = 9\*(Ts+1b\*Tb) = 9\*101 = 909.

Первая фаза займет Т4 = Т2+Т3 = 999+909 = 1908 (Т4 = Т2 + Т3 = 918 + 909 = 1827).

*2 фаза.* Отправитель рассылает максимальный приоритет девятерым процессам. Это займет Т5 = 9\*(Ts+1b\*Tb) = 9\*101 = 909.

Общее время обоих фаз займет Т6 = Т4 + Т5 = 1908 + 909 = 2817 (Т6 = Т4 + Т5 = 2736).  
Это было время одной широковещательной упорядоченной рассылки. Нам нужно десять таких. Итоговое время Т = 10\*Т6 = 28170 (Т = 10\*Т6 = 27360).

**===отлично**

1. **Имеется механизм двоичных семафоров. Опираясь на него, реализуйте P-операцию и V-операцию для общего (считающего) семафора.**

Ответ:

Int S = N;

Semaphore access = 1; // семафор для монопольного доступа к S

Semaphore wait = 1;// при помощи него мы будет реализовывать ожидание.

P(Int S) {

wait.P();

access.P();

S = S – 1;

If(S > 0) wait.V(); //если мы последним вошли в критическую секцию(S == 0) - залочили после себя всех

access.V();

}

V(Int S) {

access.P();

S++;

If(S == 1) wait.V(); //мы освобождаем единственное место - надо разлочить ожидающих, если мы освобождаем второе и далее место - значит очереди нет, никого разлочивать не надо

access.V();

}

**===отлично**

1. **Алгоритм надежных и неделимых широковещательных рассылок сообщений. Дайте оценку времени выполнения одной операции рассылки для сети из 10 ЭВМ с шинной организацией (без аппаратных возможностей широковещания), если отправитель сломался после посылки 5-го сообщения. Время старта (время «разгона» после получения доступа к шине для передачи) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:

Отправитель посылает сообщение 5 ЭВМ, после чего ломается.(5 сообщений). В качестве уникального идентификатора сообщения используется его начальный приоритет - логическое время отправления. Пусть логическое время занимает 1 байт **(ВРЕМЯ СОСТОИТ ИЗ СЧЕТЧИКА СОБЫТИЙ И НОМЕРА ПРОЦЕССА!!!)**. Процесс-отправитель посылает сообщение группе процессов (список их идентификаторов содержится в сообщении). Пусть этот идентификатор занимает 1 байт. Тогда список будет 9 байт. Если само сообщение(для простоты) занимает 1 байт, тогда отправляется 11 байт. Итого времени прошло 5\*(Ts+11\*Tb)=555

При получении этого сообщения процессы:

1. Приписывают сообщению приоритет, помечают сообщение как «недоставленное» и буферизуют его. В качестве приоритета используется временная метка (текущее логическое время).
2. Информируют отправителя о приписанном сообщению приоритете.

Каждый из пяти получателей отправляют отправителю приоритет - 1 байт(текущее логическое время). Итого 5\*(Ts+1\*Tb)=505

"Если получатель обнаружит, что он имеет сообщение с пометкой «недоставленное», отправитель которого сломался, то он для завершения выполнения протокола осуществляет следующие два шага в качестве координатора.

1. Опрашивает всех получателей о статусе этого сообщения."  
Это и изображено на схеме. (5 запросов)

Пусть сообщение с запросом статуса содержит 2 байта. Тогда 5\*(Ts+2\*Tb)=510.

Координатор получил ответы(5 ответов). 4 получателя ответили:

1. Сообщение отмечено как «недоставленное» и ему приписан такой-то приоритет. Пусть этот ответ занимает 2 байта.

А один:

1. Он не получал это сообщение. Пусть этот ответ занимает 1 байт.

Тогда имеем 4\*(Ts+2\*Tb)+Ts+Tb=509

Координатор заново начинает весь протокол с фазы 1.(8 сообщений). Список состоит из 8 процессов(8 байт)+само сообщение(1 байт)+идентификатор сообщения(1 байт). Имеем 8\*(Ts+10\*Tb)=880

Получает ответы от всех адресатов(8 ответов). Это 8\*(Ts+1\*Tb)=808. Здесь приписанный каждым получателем приоритет - 1 байт. Далее находим максимальный приоритет и отправляем его всем получателям(8 уведомлений):

Приоритет содержит 1 байт, поэтому 8\*(Ts+1\*Tb)=808.

**Просуммируем все, что мы получили:**

**555+505+510+509+880+808+808=4575**

**Должны быть объяснены длины всех сообщений (что в них содержится и сколько места занимает)**

**Вопрос – если надежность не нужна (все работает надежно), а требуется только неделимость, то алгоритм будет быстрее работать? За счет чего?**

**Да, за счет того, что не нужно отправлять список идентификаторов процессов, т.к. уже будет гарантирована надежная передача. Да и не нужно будет заново отправлять одно и то же сообщение повторно.**

**===хорошо**

1. **15 процессов, находящихся в узлах транспьютерной матрицы размером 4\*4, одновременно выдали запрос на вход в критическую секцию. Сколько времени потребуется для прохождения всеми критических секций, если используется централизованный алгоритм (координатор расположен в узле 0,0)? Время старта равно 200, время передачи байта равно 3 (Ts=200,Tb=3). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение:  
***Централизованный алгоритм.***

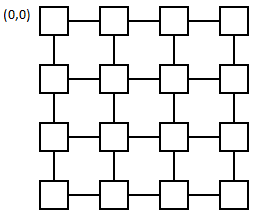
Все процессы запрашивают у координатора разрешение на вход в критическую секцию и ждут этого разрешения. Координатор обслуживает запросы в порядке поступления. Получив разрешение процесс входит в критическую секцию. При выходе из нее он сообщает об этом координатору. Количество сообщений на одно прохождение критической секции - 3.

Недостатки алгоритма - обычные недостатки централизованного алгоритма (крах координатора или его перегрузка сообщениями).

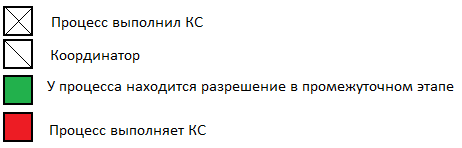
Отсюда 15\*3 (Ts+Tb\*L)

А **что такое L -** **Должны быть объяснены длины всех сообщений (что в них содержится и сколько места занимает)**

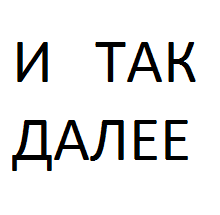
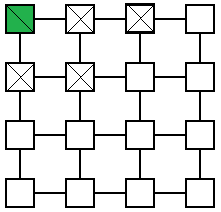
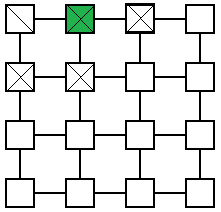
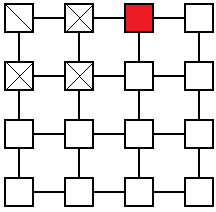
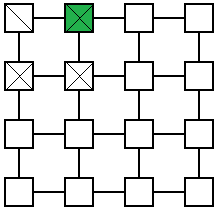
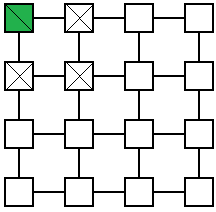
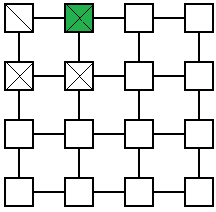
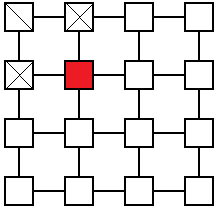
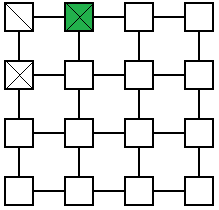
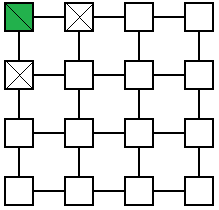
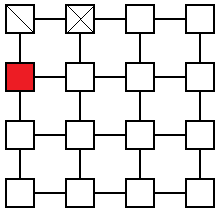
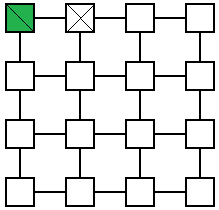
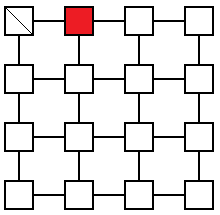
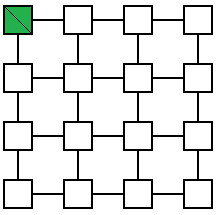
Решение было не совсем правильным. Вышеуказанное решение для шинной организации, а у нас транспьютер. Как примерно будет проходит процесс лучше всего отразят следующие картинки.



Введем следующие обозначения. Зеленый квадрат означает, что «разрешение идет» от координатора к запросившему процессу (от запросившего процесса к координатору), и «разрешение» должно быть передано дальше.



Принцип такой (порядок – слева направо, сверху вниз):



Таким образом, продолжая по данному принципу, получаем 96 посылок сообщений. Пусть размер одного сообщения 1 байт (т.к. содержится только разрешение).

Ответ: Общее время Т = 96\*(Ts+1b\*Tb) = 96\*(203) = 19488

**===отлично**

1. **Причинная консистентность памяти и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует модификация 10 различных переменных, если все 10 процессов (каждый процесс модифицирует одну переменную), находящихся на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания), одновременно выдали запрос на модификацию своей переменной. Время старта (время «разгона» после получения доступа к шине) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи операции посылки сообщения (при одновременно выданных операциях - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми. Никаких сведений от компилятора о причинной зависимости операций модификации не имеется.**

Решение:

1.Последовательность операций записи, которые потенциально причинно зависимы, должна наблюдаться всеми процессами системы одинаково, параллельные операции записи могут наблюдаться разными узлами в разном порядке.

2. Из условия (см. конец задачи) ясно, что процессы не знают о причинных зависимостях между переменными. И задача сводится к задаче 5.

**Нет, не сводится - не знают о причинных зависимостях, но могут определить потенциально причинные зависимости и реализовать эту модель эффективнее, чем последовательную!**

Реализация причинной консистентности может осуществляться следующим образом:

* все модификации переменных на каждом процессоре нумеруются;
* всем процессорам вместе со значением модифицируемой переменной рассылается номер этой модификации на данном процессоре, а также номера модификаций всех процессоров, известных данному процессору к этому моменту;
* выполнение любой модификации на каждом процессоре задерживается до тех пор, пока он не получит и не выполнит все те модификации других процессоров, о которых было известно процессору - автору задерживаемой модификации.

Ответы по плану.

1) Последовательность операций записи, которые потенциально причинно зависимы, должна наблюдаться всеми процессами системы одинаково, параллельные операции записи могут наблюдаться разными узлами в разном порядке

2) а) при записи рассылается всем;

б) читается из локальной памяти;

в) при записи рассылается всем;

г) нет

3) Оценка времени. Например, первый процесс рассылает всем модификации, его модификация станет известна всем. Потом второй процесс в сообщении, помимо модификации, прикрепляет что ему известна модификация первого процесса. Третьему известны модификации первого и второго процессов и т.д. Каждый процесс рассылает по 9 сообщений. Пусть под адрес и значение переменной отведем по 1 байту. Под номер модификации тоже 1 байт. Под номер процессора, которому принадлежит модификация тоже 1 байт. Тогда размер сообщения 1 процесса 2 байта, второго – 4, третьего – 6 и т.д.

Размер десятого – 20 байтов. Соответственно времена:

Т1 = 9\*(Ts+2b\*Tb)

T2 = 9\*(Ts+4b\*Tb)

………..

T10 = 9\*(Ts+20b\*Tb)

Общее время Т = Т1 +….+ Т10 = 9\*(10\*Ts + 110b\*Tb) = 9\*(1000+110) = 9\*1100 = 9900

**===отлично**

1. **Имеется команда TSL и команда объявления прерывания указанному процессору. Опираясь на него, реализуйте на мультипроцессоре P-операцию и V-операцию для двоичного семафора.**

Решение: Tsl(r, s) делает [r = s, s = 1] – это неделимая операция.

Enter\_region: //P-operation

Tsl reg, flag

Cmp reg, #0

Je Go

<Ждем прерывание>

Go: Ret

Leave\_region:// V-operation

Move flag, #0

<Отправляем сигнал> **КОМУ? Это прерывание указанному процессору?**

Ret

**Да, одному из заблокированных процессоров. Можно дополнительно ввести список ждущих, и отправлять первому из них прерывание. Тогда**

Je Go Je Go

<Ждем прерывание> изменится на <Добавляем себя в список ждущих>

Go: Ret <Ждем прерывание>

<Убираем себя из списка>

Go: Ret

**На одном процессоре может быть много процессов – активных и заблокированных (например, по ожиданию семафора). Команда TSL нужна для взаимного исключения, чтобы операционные системы одновременно не стали работать с очередями процессов и заначениями семафоров. При освобождении семафора разблокируется первый процесс из очереди ожидающих. Это происходит на том процессоре, на котором выполнена операция освобождения, и надо известить другие процессоры (один конкретный, на котором должен выполняться разблокированный процесс, или все другие) – для этого команда прерывания.**

===

1. **В транспьютерной матрице размером 4\*4, в каждом узле которой находится один процесс, необходимо переслать сообщение длиной L байт из узла с координатами (0,0) в узел с координатами (3,3). Сколько времени потребуется для этого при использовании а) неблокирующих и б) блокирующих операций MPI? Время старта равно 300, время передачи байта равно 1 (Ts=300,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

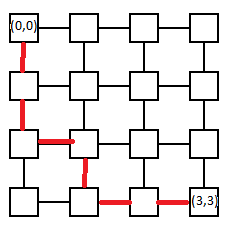
Решение:

При неблокирующих и блокирующих операциях время передачи одинаковое.

Передавать можно с использованием конвейера или без конвейера.

Без конвейера.

Потребуется 6 посылок по L байт. Время Т = 6\*(Ts+L\*Tb)



С конвейером.

Идея: разделить исходное сообщение на К частей.

Время передачи первой части 6\*(Ts+Tb\*L/K).

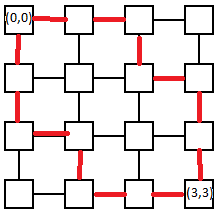
Время передачи остальных частей (K-1)(Ts+Tb\*L/K).

Время Т = (K+5)(Ts+Tb\*L/K), К1 = [sqrt(5\*Tb\*L/Ts)], К2 = К1+1.

K = К1 или К2, в зависимости где достигается минимум времени.

Если исходное сообщение разделить на 2, и каждое полученное сообщение разделив на К частей, пустить по 2 параллельным маршрутам. То получим T = (K+5)(Ts+Tb\*(L/2)/K).  
К1 = [sqrt(5\*Tb\*(L/2)\*Ts)], К2 = К1+1.

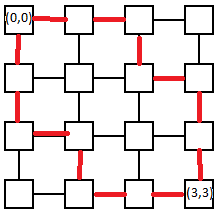
K = К1 или К2, в зависимости где достигается минимум времени.



**А без конвейера нельзя использовать несколько маршрутов?**

**===хорошо**

Можно, конечно.  
Потребуется 6 посылок по L/2 байт по каждому из двух маршрутов, например, следующего вида.



В этом случае время будет Т = 6\*(Ts+(L/2)\*Tb)

1. **Все 10 процессов, находящихся на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания), одновременно выдали запрос на вход в критическую секцию. Сколько времени потребуется для прохождения всеми критических секций, если используется децентрализованный алгоритм с временными метками. Доступ к шине ЭВМ получают последовательно в порядке выдачи операции посылки сообщения (при одновременно выданных операциях - в порядке номеров ЭВМ). Время старта равно 100, время передачи байта равно 3 (Ts=100,Tb=3). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение: Каждый процесс шлет запрос на вход в критическую секцию всем остальным 9 процессам. Итого: 9\*10\*(Ts+Tb\*L)

После выполнения критической секции процесс смотрит свои запросы на вход в своем списке и шлет им Ok(Ok может отправляться и до входа в критическую секцию).

В результате на посылку Ok уходит 9\*10\*(Ts+Tb\*Lok). **ЧТО ТАКОЕ Lok?** При этом мы считаем, что время прохождения КС=0. Пусть L=Lok

Итого: 2\*9\*10\*(Ts+Tb\*L)

**Надо задавать размеры в байтах!**

**В ТРЕБОВАНИЯХ написано:**

3. Длины сообщений должны быть объяснены. Пример: “предположим, что адрес переменной занимает 4 байта, а значение - 8 байтов. Тогда длина сообщения равна 120 байтов”. Это нужно для того, чтобы видеть, как Вы поняли алгоритм - что посылается в каждом сообщении (например, что такое логическое время, как идентифицируется модифицируемая переменная и т.п.)

**Решение:  
*Децентрализованный алгоритм на основе временных меток.***

Требуется глобальное упорядочение всех событий в системе по времени. Для целей упорядочения всех событий удобно потребовать, чтобы их времена никогда не совпадали. Это можно сделать, добавляя в качестве дробной части к времени уникальный номер процесса (40.1, 40.2) **И ЭТО 1 БАЙТ???**.

***Вход в критическую секцию***

Когда процесс желает войти в критическую секцию, он посылает всем процессам сообщение-запрос, содержащее имя критической секции, номер процесса и текущее время.

После посылки запроса процесс ждет, пока все дадут ему разрешение. После получения от всех разрешения, он входит в критическую секцию.

***Поведение процесса при приеме запроса***

Когда процесс получает сообщение-запрос, в зависимости от своего состояния по отношению к указанной критической секции он действует одним из следующих способов.

1. Если получатель не находится внутри критической секции и не запрашивал разрешение на вход в нее, то он посылает отправителю сообщение «OK».
2. Если получатель находится внутри критической секции, то он не отвечает, а запоминает запрос.
3. Если получатель выдал запрос на вхождение в эту секцию, но еще не вошел в нее, то он сравнивает временные метки своего запроса и чужого. Побеждает тот, чья метка меньше. Если чужой запрос победил, то процесс посылает сообщение «OK». Если у чужого запроса метка больше, то ответ не посылается, а чужой запрос запоминается.

***Выход из критической секции***

После выхода из секции он посылает сообщение «OK» всем процессам, запросы от которых он запомнил, а затем стирает все запомненные запросы.

Количество сообщений на одно прохождение секции - 2(n-1), где n - число процессов.

Кроме того, одна критическая точка заменилась на n точек (если какой-то процесс перестанет функционировать, то отсутствие разрешения от него всех остановит).

В сообщении-запрос содержится номер КС, номер процесса, и текущее время. Отведем по 1 байту каждому, получим размер сообщения – 3 байта. Пусть «ОК» занимает 1 байт.

Перед заходом в КС процесс должен всем отправить запрос и от всех получить «ОК».

Время одного прохождения КС Т1 = 9\*(Ts+3b\*Tb) + 9\* (Ts+1b\*Tb) = 9\*109+9\*103= 1908. Количество КС = 10.

Ответ: Общее время Т = 10\*Т1 = 19080.

===отлично