Тема 1.

(См. Лекция 1: "Введение в параллельные и распределенные системы ")

Какие аппаратные механизмы необходимы для организации мультипрограммного режима? Как обеспечить мультипрограммный режим без этих механизмов? Как обеспечить, если отсутствует только один из них?

Решение.

- Аппаратные средства:
    1. Система прерываний
    2. Механизм защиты ОП
    3. Привелегированный режим
    4. Таймер
- Если нет этих механизмов, то их можно имитировать программно, но это очень неэффективно. Также можно использовать специализированные компиляторы.
- Отсутствие одного из 4х пунктов-равносильно отсутствию всех.

Тема 2.

(См. Лекция 2: "Операционные системы мультипроцессорных ЭВМ")

1. Если в алгоритме Деккера не изменять значение переменной ***turn*** при выходе из критической секции, то каким требованиям он перестанет удовлетворять? Объясните, почему.

Решение. Он перестанет удовлетворять 3-ему требованию: *"ни один процесс не должен бесконечно долго ждать разрешения на вход в критический интервал (если ни один процесс не будет находиться внутри критического интервала бесконечно)"* .

Один и тот же процесс легко может войти в критическую секцию два раза подряд. Кроме этого, один процесс может зависнуть в активном ожидании на операторе while (turn != i);

Переменная **turn** определяет номер процесса, который получит доступ к критической секции, в случае, когда сразу два процесса попытались в нее войти. Но не более того.

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

Решение.Тут в лекциях допущена ошибка в описании функции P семафора. Наверно, достаточно завести **два двоичных семафора** и **int переменную** для имитации ей считающего:

Возможно, эта версия хорошая.

```
int count = N;

boolSemaphore sem = 1;


P(s) {

    while(true) {

        P(sem);

        if(count > 0) {

            count--;

            V(sem);

            break;

        } else {

            V(sem);
```

```
        }

    }

}



V(s) {

    count++;

}
```

Еще есть такая версия

```
semaphore access = 1;
semaphore wait = 0;
int s = <число процессов, которые могут находиться в КС>;

P(s) {
  P(access);
  if s <= 0 { s = s - 1; V(access); P(wait); }
  else { s = s - 1; V(access); }
}

V(s) {
  P(access);
  if s < 0 { V(wait); }
  s = s + 1;
  V(access);
}
```

3. Имеется механизм двоичных семафоров. Опираясь на него, реализуйте операторы POST(имя переменной-события) и WAIT(имя переменной-события).

Решение.В этом решении возможны ошибки в коде.Наверно, достаточно завести **один двоичный семафор** и **bool переменную** для обозначения самого события (произошло - не произошло) все.

```
boolSemaphore pSem = 1;
boolSemaphore wSem = 1;
bool eventHappened = false;

POST(event) {
    P(pSem);
    if (eventHappened == false) {
        eventHappened = true;
        <разблокировать все ожидающие процессы>
    }
    V(pSem);
}
```

```
WAIT(event) {
     P(wSem);
     if (eventHappened == false) {
          <блокируем текущий процесс>
     }
     V(wSem);
}
```

Комментарии: семафор мы используем для того, чтобы не было ситуаций вроде:

- Несколько одновременных вызовов POST
- WAIT уже вошел в **if** и собрался блокироваться, а в это время POST *полностью выполнился*, однако в очереди на разблокировку еще не было WAIT-а. После этого WAIT вносит себя в очередь. Результат - событие объявлено, а процесс его ждет.

  [Предлагается следующий код:

  bsemaphore sem = 0;
  WAIT(event) { P(sem); V(sem); }
  POST(event) { V(sem); }]

Вариант кода:

```
void WAIT() {
   while(clearing); // подождать, если идет очистка
   if (posting)
      return; // событие объявлено - можно работать
   else {
      waiting++; // событие не объявлено - записаться в ждущие
      P(wait); // ждать :)
   }
}
```

```
void POST() {
   if(posting)
      return; // уже объявлено - ничего не надо делать
   posting = true; // объявляем
   while(waiting > 0) { // разблокируем ждущие процессы
      waiting--;
      V(wait);
   }
}
```

```
void CLEAR() {
   if(!posting) // не объявлено - ничего не делаем
      return;
   clearing = true; // начать очистку
   posting = false; // очистка - дожидаемся пока POST освободит процессы, которые ждали до CLEAR.
   while(waiting > 0); // подождать пока POST освободит "старые" процессы
   clearing = false; // закончить очистку
}
```

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

Решение.
Пусть операция TSL работает с bool переменными и ее выполнение не может быть прервано:

```
bool TSL(bool lock) {
    boolean initial = lock;
    lock = true;
    return initial;
}
```

Используют ее следующим образом:

```
boolean lock = false
void Critical() {
    while TSL(lock)
        skip //spin until lock is acquired
    critical section //only one process can be in this section at a time
    lock = false //release lock when finished with the critical section
}
```

Попробуем реализовать P (занять) и V (освободить) операции для двоичного семафора с помощью этой штуки.

```
bool forLock = false;
bool forSemaphore = false;

P(s) {
    while TSL(forLock) ;
    if (forSemaphore == true) <прерыванием остановить текущий процесс>;
    forSemaphore = true;
    forLock = false;
}

V(s) {
    while TSL(forLock) ;
    if (length == 0) <прерыванием запустить 1 ожидающий процесс>;
    forSemaphore = false;
    forLock = false;
}
```

<hr>

По-моему, усложняете... ИМХО, можно так:

bool loc = false;
bool glob = false;

P(S) {
do
tsl(loc, glob)

until (loc == false);
}

V(S) {
glob = false;
}

5. Правильно ли использованы события в алгоритме, который реализует метод верхней релаксации? Оцените, насколько этот алгоритм можно выполнить быстрее, чем последовательный, если число процессоров мультипроцессора = N, время выполнения одного оператора присваивания (*A[i][j]*=….) равно 1, временами выполнения остальных операторов можно пренебречь.

```
float  A[ L1 ][ L2 ];
struct condition s[ L1 ][ L2 ];

for ( i = 0; i < L1; i++)
    for ( j = 0; j < L2; j++)
        { clear( s[ i ][ j ]) }

for ( j = 0; j < L2; j++)
    { post( s[ 0 ][ j ]) }

parfor ( i = 1; i < L1-1; i++)
        for ( j = 1; j < L2-1; j++)
    {
        wait( s[ i-1 ][ j ]);
        A[ i ][ j ] = (A[ i-1 ][ j ] + A[ i+1][ j ] + A[ i ][ j-1 ] + A[ i ]
[ j+1 ]) / 4;
        post( s[ i ][ j ]);
    }
```

Решение.

Parfor is a parallel loop construct that executes the iterations as separate threads, while the body of each iteration is sequential. The loop is unravelled by assigning a constant value for the loop control variable within each iteration. The prequisite for this is that the values of the loop variable must be obtainable without executing the body of any iteration.

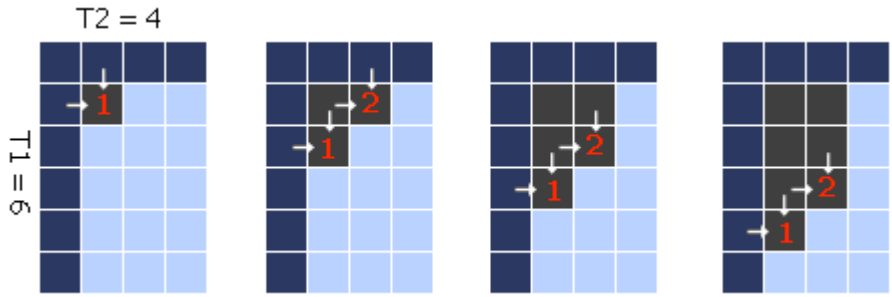
Parfor statement completes only when all the iterations have completed, but as with par-statement nothing can be said about the order of execution of different threads. It is only quaranteed that when the program execution proceeds past the parfor block all the threads inside the parfor have finished.

По поводу правильности - если сравнить этот код с тем, что в лекциях, то вот что имеем (красным отмечен пропущенный в задаче код):

```
float  A[ L1 ][ L2 ];
struct condition s[ L1 ][ L2 ];

for ( i = 0; i < L1; i++)
    for ( j = 0; j < L2; j++)
        { clear( s[ i ][ j ]) }

for ( i = 0; i < L1; i++)
    { post( s[ i ][ 0 ]) }

for ( j = 0; j < L2; j++)
```

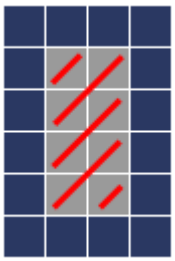
```
    { post( s[ 0 ][ j ]) }

parfor ( i = 1; i < L1-1; i++)
for ( j = 1; j < L2-1; j++)
    {
        wait( s[ i ][ j-1 ]);
        wait( s[ i-1 ][ j ]);
        A[ i ][ j ] = (A[ i-1 ][ j ] + A[ i+1][ j ] + A[ i ][ j-1 ] + A[ i ]
[ j+1 ]) / 4;
        post( s[ i ][ j ]);
    }
```

Быстрее его можно выполнить, если выполнять действия A[]... на разных процессорах, см. картинку. Цифры -

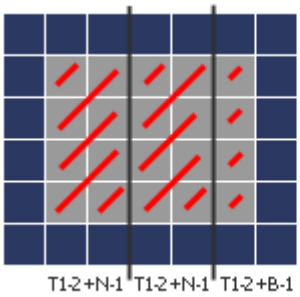


номер процессора.
Рассмотрим случай, когда T1 >= T2 (Другой - очевидно симметричен).

1. N >= T2 - 2 (N - число работающих процессоров) Максимальная скорость, как на картинке. По сути нужно столько тактов, сколько диагоналей проходит через нужную нам область (прямоугольник за вычетом границы толщиной 1). Это: (T1 - 2) + (T2 - 3).



2. N < T2 - 2. Тут лучше привести задачу к виду 1, разделив на несколько. Пусть A = T2 - 2 / N, B = T2 - 2 % N (частное и остаток), тогда можно выполнить эту задачу как последовательность из A + 1 задач . Получим время выполнения: A * (T1 - 2 + N - 1) + (T1 - 2 + B - 1)



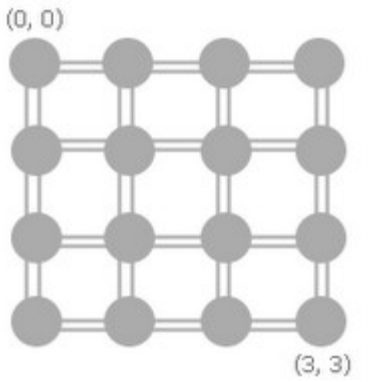
T1 = 6
T2 = 7
N = 2

T2 - 2 = 2 * N + 1

Тема 3.

(См. Лекция 3: "Коммуникации в распределенных системах")

Главная мысль из этого флуда:"Задача найти ОДНО ИЗ решений, не занимайтесь поиском оптимального, просто скажите, какие есть методы оптимизации (конвейер, разделение и т.п.)"



1. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию барьер (MPI_BARRIER) для всех процессов. Сколько времени потребуется для этого, если все процессы выдали ее одновременно. Время старта равно Ts, время передачи байта равно Tb (Ts=10,Tb=2). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение. В лекциях про MPI_BARRIER почти ничего нет, вот для информации:
Точки синхронизации, они же барьеры.

Этим занимается всего одна функция:

int MPI_Barrier(MPI_Comm comm);

MPI_Barrier останавливает выполнение вызвавшей ее задачи до тех пор, пока не будет вызвана изо всех остальных задач, подсоединенных к указываемому коммуникатору. Гарантирует, что к выполнению следующей за MPI_Barrier инструкции каждая задача приступит одновременно с остальными.

Это единственная в MPI функция, вызовами которой гарантированно синхронизируется во времени выполнение различных ветвей! Некоторые другие коллективные функции в зависимости от реализации могут обладать, а могут и не обладать свойством одновременно возвращать управление всем ветвям; но для них это свойство является побочным и необязательным - если Вам нужна синхронность, используйте только MPI_Barrier.
Пусть координатор находится в точке (0, 0). В один момент времени все 15 оставшихся процессов вызывают **MPI_BARRIER**. Это выражается в отправке в сторону Координатора 1 байта каждым. Составим таблицу:

Расстояние до координатора	**1**	**2**	**3**	**4**	**5**	**6**
Количество каналов в сторону координатора	**2**	4	6	6	4	2
Количество байт для передачи	2	3	4	3	2	1

Задержка при передаче этого кол-ва байт по этому кол-ву каналов	1	2	2	2	1	1

Красным-bold отмечено самое узкое место в топологии каналов в нашем случае. Задержка получается делением кол-ва байт для передачи на кол-во каналов в сторону координатора в худшем случае (а это 2). В итоге имеем задержку байта, посланного от **(3, 3)**: 6 * Ts + (1 + 2 + 2 + 2 + 1 + 1) * Tb = 6 * Ts + 9 * Tb

После этого он начнет рассылать всем процессам сообщение о том, что барьер создан (работа синхронизирована, все дошли до барьера).

Расстояние от координатора	**0**	**1**	**2**	**3**	**4**	**5**
Количество каналов от координатора	2	4	6	6	4	2
Количество байт для передачи	15	13	10	6	3	1
Задержка при передаче этого кол-ва байт по этому кол-ву каналов	8	7	5	3	2	1

Кроме этого, есть еще 6 затрат на установление соединений (реально соединений было больше, просто многие шли параллельно). В сумме 6 * Ts + 26 * Tb.

Ответ: 12 * Ts + 35 * Tb

[Рассылка всем процессам какого-то сообщения это MPI_BCAST (см. следующую задачу).
У нее ответ: 6 * (Ts + 1 * Tb.)] - да, тут вообще все можно по разному обыграть. Главное обосновать.

2. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию передачи сообщения длиной N байт всем процессам от одного (MPI_BCAST) - процесса с координатами (0,0). Сколько времени потребуется для этого, если все процессы выдали ее одновременно. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

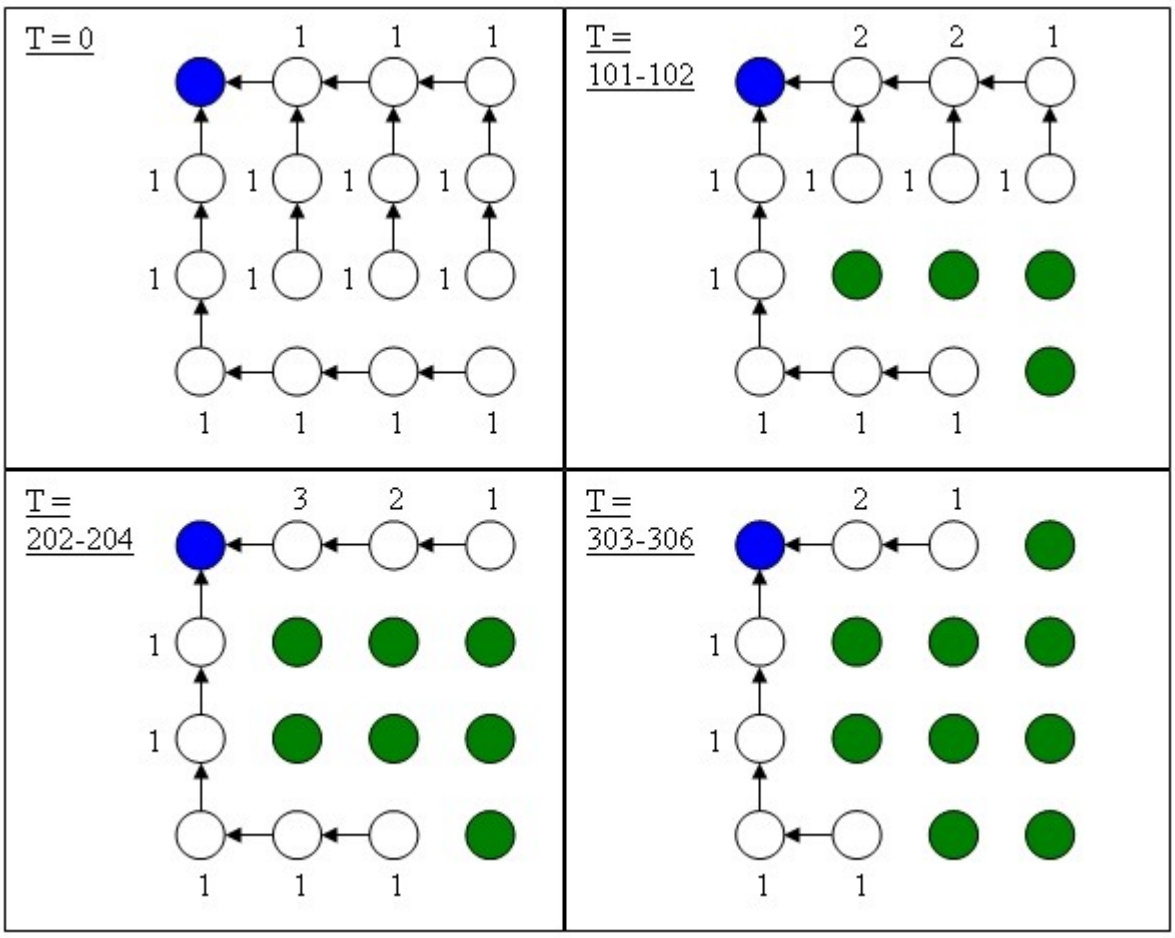
Решение.Если кто понял, что значит фраза *"... если все процессы выдали ее одновременно", скажите.* По-моему это неправильный копи-паст задач просто.

MPI_BCAST - это широковещательная рассылка сообщения всем остальным процессам в группе.

Предположим, что нам из точки (0, 0) достаточно послать сообщения только своим соседям, а они сами передадут его копии дальше. Тогда время будет равно времени пути сообщения до самой дальней точки, а это 6 переходов.

Ответ: 6 * (Ts + 1 * Tb). Только не 1*Tb, а N*Tb. Потому что N байт.

3. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию сбора данных от всех процессов (длиной один байт) для одного (MPI_GATHER) - процесса с координатами (0,0). Сколько времени потребуется для этого, если все процессы выдали ее одновременно. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми. **Решение.**



Цифрами обозначено количество байт, которое транспьютер хочет отправить в данный момент времени.

Итого получаем 6 * (Ts + Tb)

А как работает gather, разве процессоры сначала не должны узнать о том ,что (0,0) хочет от них информацию? Т.е не надо считать время, которое потратится сначала на извещение? И не понятно откуда в такой картинке 6Ts +… . Вот последний рисунок, это время 3Ts +…, при этом левая ветвь состоит из 4 посылок, которые не сделанны.

Согласен, надо подправить.

4. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию рассылки данных (длиной один байт) всем процессам от одного (MPI_SCATTER) - процесса с координатами (0,0). Сколько времени потребуется для этого, если все процессы выдали ее одновременно. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

5. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию суммирования 16 чисел (каждый процесс имеет свое число). Сколько времени потребуется

для получения всеми суммы, если все процессы выдали эту операцию редукции одновременно? А сколько времени потребуется для суммирования 64 чисел в матрице 8*8? Время старта равно единице, время передачи байта равно нулю (Ts=1,Tb=0). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

6. 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 единице, время передачи байта равно нулю (Ts=1,Tb=0). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

7. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо переслать очень длинное сообщение (длиной L байт) из узла с координатами (0,0) в узел с координатами (3,3). Сколько времени потребуется для этого. А сколько времени потребуется для пересылки из узла с координатами (1,1) в узел с координатами (2,2). Время старта равно времени передачи байта (Ts=Tb). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

8. 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, если передача сообщений выполняется в буферизуемом режиме MPI? А сколько времени потребуется при использовании синхронного режима и режима готовности? Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

- **MPI_BSEND** - передача сообщения *с буферизацией*. Последовательность вызова и окончания операций вызова и приема произвольна, операция **send** завершается тогда, когда сообщение изъято из памяти и помещено в буфер. Если места в буфере нет - ошибка программы. Операция локальная.
- **MPI_SSEND** - передача сообщения *с синхронизацией*. Последовательность вызова и завершения операций вызова и приема произвольна, но операция **MPI_SSEND** завершается только после начала выполнения операции receive. Операция нелокальная.
- **MPI_RSEND** - передача сообщения *по готовности*. Операция **MPI_RSEND** просто посылает данные на удаленный компьютер и сразу возвращает управление программе. Что с ними дальше происходит, мы можем не знать, если их там не ждали то будет ошибка. Операция локальная.

Решение.

9. 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 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

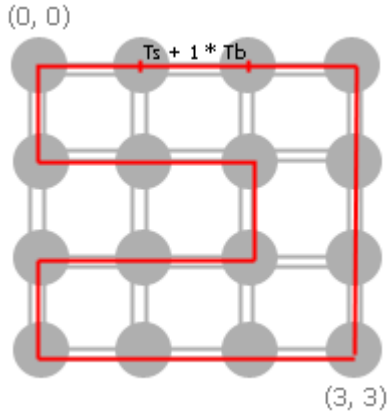
Решение.

Тема 4

(См. Лекция 4: "Синхронизация в распределенных системах")

1. Все 16 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 круговой маркерный алгоритм. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

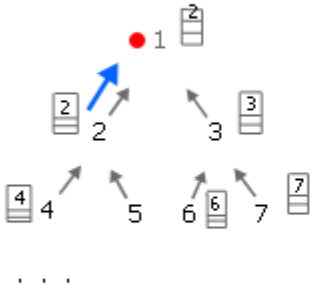
Решение.
Все процессы составляют логическое кольцо, когда каждый знает, кто следует за ним. По кольцу циркулирует маркер, дающий право на вход в критическую секцию. Получив маркер (посредством сообщения точка-точка) процесс либо входит в критическую секцию (если он ждал разрешения) либо переправляет маркер дальше. После выхода из критической секции маркер переправляется дальше, повторный вход в секцию при том же маркере не разрешается.



Ответ: 15 * (Ts + 1 * Tb)

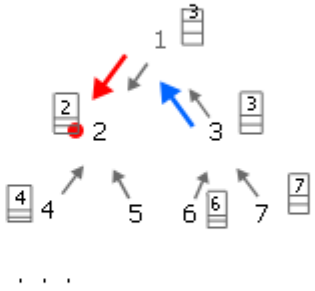
2. Все 16 процессов, находящихся на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания), одновременно выдали запрос на вход в критическую секцию. Сколько времени потребуется для прохождения всеми критических секций, если используется древовидный маркерный алгоритм. Время 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.



Процесс 1 обладает маркером, выполнил КС, процесс 2 получил шину и отправил ЗАПРОС в сторону маркера, 1 получил ЗАПРОС от 2

Время: Ts + 1 * Tb



Процесс 1 получает шину, отправляет маркер автору первого запроса из очереди. 2 получает его и выполняет КС. 3 получает шину и отправляет ЗАПРОС к 1.

Время: 2 * (Ts + 1 * Tb)

Я так понял, что вершины получат маркер в порядке их номеров. Количество ЗАПРОС-ов и МАКЕР-ов будет равно удвоенной длине пути между вершинами (соседними).

Если посчитать вручную нужное количество запросов и передач маркеров, то когда 5-ая вершина будет выполнять свою КС их будет 18, на 8-ой - 40, а на 6 - 102.

Ответ: 102 * (Ts + 1 * Tb).

[В нулевой момент времени к шине поступают запросы от устройств 2-16. Первые 15*(Ts + 1 * Tb) тактов будут отправляться ЗАПРОСы от устройств 2-16. На рисунках же обозначена последовательность получения доступа к шине следующая: 2-е, 1-е, 3-е и т.д. устройства.

Другой вопрос заключается в том, что последним устройством на 4-м уровне дерева будет 15-е устройство. Тогда куда ни присоединишь 16-е, дерево будет не сбалансированным.] - [Про нулевой момент - можно поспорить. Ведь у нас все бесконечно быстрое, кроме передач по сети, а шину получает процесс с меньшим номером. Значит, пока 1-ому есть, что передавать, он будет передавать. По-моему это самая простая из допустимых интерпретация, т.к. в других случаях получаем взрыв мозга :) А про дерево - куда бы ни присоединить 16-ый - оно будет сбалансированным. У сбалансированного дерева нижний уровень не должен быть до конца заполнен.]

3. Все 16 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 децентрализованный алгоритм с временными метками. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.Вообще, для случая равных временных меток в алгоритме ничего не сказано (он зависает), даже на Википедии. Поэтому доопределим его сами, например при равных временных метках выигрывает процесс с меньшим номером. Также не будем учитывать размера ОК и ЗАПРОСа в байтах.

Предположим, что процессы за один сеанс передачи могут передать все скопившиеся у них сообщения в данную сторону. Тогда первому процессу для получения согласия от всех потребуется ждать 12 шагов (время отзыва самого дальнего процесса, 6 передач ЗАПРОСа и 6 передач ОК). За это время все успеют все передать и останутся

стоять в очередях лишь отложенные запросы. Поэтому когда 1-ый выполнит КС, то он передаст ОК за 1 шаг.

Далее по аналогии, можно дать номера вершинам таким образом (змейкой), что ОК будет передаваться все время на расстояние 1. Итого - 15 передач ЩЛ, 15 * Ts.

Ответ: 15 * Ts + 12 * Ts

[Временная метка и маркер - это совершенно разные понятия.

Алгоритм временных меток:
1. Рассылка всем процессам сообщение-запрос о входе в КС.
2. Ожидание приема подтверждений.

Порядок входа в КС (змейкой или не змейкой) здесь абсолютно не важен. Временная метка -- это просто локальное время.Оно есть на каждом устройстве. Поэтому выражение "отдаст метку" лишено всякого смысла.]
- [Спасибо, исправил. Про порядок - не совсем, в этом алгоритме используется глобальное время (см. лекции). Поэтому с учетом доопределения алгоритма он важен.]

4. Все 16 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 широковещательный маркерный алгоритм. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение. Время работы зависит от того, в каком порядке запросы будут занесены в очередь, которая находится в самом маркере. При обычной (слева-направо, сверху-вниз) нумерации процессов и нахождении маркера изначально в 1-ом процессе, запросы будут внесены в маркер практически по возрастанию (вроде бы). Предположим, что это действительно так. Тогда маркер будет переходить по матрице в порядке номеров вершин.

Пусть длинна маркера в битах - A, тогда проход маркера по всем вершинам займет 1 + 1 + 1 + 4 (3 перехода по верхней строке + один переход на новую строчку) - так 3 раза + еще 3 перехода, итого - 24 перехода. Время - 24 * (Ts + A * Tb). Размер маркера может быть около 16 (список номеров запросов) + 16 (очередь запросов) байт. Плюс время передачи запроса от 2-го к 1-му (в самом начале, B - размер запроса).

Ответ: 24 * (Ts + A * Tb) + 1 * (Ts + B * Tb)

[В ответе не учтены отправки широковещательных сообщений «ЗАПРОС»:

Вход в критическую секцию

1) Если процесс Pk, запрашивающий критическую секцию, не имеет маркера, то он увеличивает порядковый номер своих запросов RNk[k] и посылает широковещательно сообщение «ЗАПРОС», содержащее номер процесса (k) и номер запроса (Sn = RNk[k]).

] - [Вроде теперь учтены]

5. 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 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.Пусть для одновременно пришедших запросов координатор отдает предпочтение процессу с меньшим номером, и все запросы ставит в очередь. Тогда 2 процесс начнет выполняться уже через 1 (запрос) + 1 (разрешение) передачи сообщений, 3-ий - через 1 (2-ой сказал К что закончил) + 2 (К разрешил 3-ему работать), и т.д.

В общем виде - путь от предыдущего узла до Координатора + путь от координатора до текущего узла. В итоге имеем:
[0 + 1+1 + 1+2 + 2+3] +
[3+1 + 1+2 + 2+3 + 3+4] +
[4+2 + 2+3 + 3+4 + 4+5] +
[5+3 + 3+4 + 4+5 + 5+6]
= 91 обмен сообщениями.

Если принять что в каждом запросе было по 1 байту, то получил 91 * (Ts + 1 * Tb).

Ответ: 91 * (Ts + 1 * Tb).

6. Сколько времени потребует выбор координатора среди 16 процессов, находящихся в узлах транспьютерной матрицы размером 4*4, если используется алгоритм задиры? Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми. Задира расположен в узле с координатами (0,0) и имеет уникальный номер 0.

Решение.
Если процесс обнаружит, что координатор очень долго не отвечает, то инициирует выборы. Процесс P проводит выборы следующим образом:

1. P посылает сообщение "ВЫБОРЫ" всем процессам с большими чем у него номерами.
2. 2. Если нет ни одного ответа, то P считается победителем и становится координатором.
3. 3. Если один из процессов с большим номером ответит, то он берет на себя проведение выборов. Участие процесса P в выборах заканчивается.

В любой момент процесс может получить сообщение "ВЫБОРЫ" от одного из коллег с меньшим номером. В этом случае он посылает ответ "OK", чтобы сообщить, что он жив и берет проведение выборов на себя, а затем начинает выборы (если к этому моменту он уже их не вел). Следовательно, все процессы прекратят выборы, кроме одного - нового координатора. Он извещает всех о своей победе и вступлении в должность сообщением "КООРДИНАТОР".

Во втором пункте алгоритма нужно убедиться, что нет ни одного ответа - пусть для этого нужно будет подождать время Timeout.

ВЫБОРЫ от 0 до 15-го процесса доходят за 6 шагов, он ждет Timeout и дальше за 6 шагов всех оповещает о новом координаторе. Все вместе это занимает 6 * (Ts + A * Tb) + 6 * (Ts + 1 * Tb) + Timeout времени, где A - среднее количество байт для передачи в начальные моменты, его мне оценить не удалось, поэтому лучше вообще пренебречь Tb (тем более что Tb << Timeout).

Ответ: 12 * Ts + Timeout

[Забавность ситуации состоит в том, что задира имеет самый меньший из всех транспьютеров уникальный номер 0. Вот уж кто точно не станет координатором.] - [Если все остальные скончались - то станет]

Мне кажется, не очень здорово вводить параметр Timeout - этим мы делаем предположение о скорости работы процессов, что в лекциях не рекомендуется.

Попробую предложить другое решение. Пусть передается сообщение между процессами в двух соседних узлах. Если номер отправителя меньше, то получатель первым делом шлет ему "ОК", продолжает выборы, пересылая всем остальным уже свой номер. Если же номер отправителя больше, получатель понимает, что не быть ему координатором, ничего не возвращает отправителю, всем остальным рассылает сообщение от имени отправителя с уже его номером. [Не знаю, можно ли так делать, но выглядит оптимальнее, чем один будет еще и еще раз слать все всем остальным].

Теперь, по условию, у нас решетка 4x4, сообщения идут из 0x0 "по слоям" решетки в сторону 3x3. Наихудший случай будет, если на каждом следующем слое номер процесса будет больше, чем на предыдущем (т.е. выбираемый координатор находится в 3x3). Тогда каждому процессу (на всех слоях, кроме первого и последнего) надо будет потратить 2*(Ts + N*Tb) для информирования предыдущего слоя и начала выборов для следующего. Тогда общий ответ 6*2*(Ts + N*Tb).

Если же выбираемый координатор находится где-то еще, а не в 3x3, то просто всем процессам на следующих слоях не надо пересылать сообщения обратно, что ускорит процесс.

7. Сколько времени потребует выбор координатора среди 16 процессов, находящихся в узлах транспьютерной матрицы размером 4*4, если используется круговой алгоритм? Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.
Алгоритм основан на использовании кольца (физического или логического), но без маркера. 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.
Предположим, что у нас в логическом кольце N живых процессов. Заметим также, что отправка подтверждений сообщений не отнимают у всего алгоритма времени.

1-ый этап - рассылка сообщения ВЫБОРЫ (длинны A). Она происходит за N шагов.

2-ой этап - рассылка КООРДИНАТОР (длинны B), это тоже N шагов. В сумме - N * (Ts + A * Tb) + N * (Ts + B * Tb).

Ответ: N * (2 * Ts + (A+B) * Tb)

[Ответ выписан с ошибкой :)
Отмечу, что для матрицы 4x4 в простом случае можно взять N=16, A=B=3 байта (1 байт - тип сообщения + 16 бит - битовый массив участников кольца). Если же не все процессы живы (что в данной задаче, похоже, не подразумевается), то нужно еще учитывать Timeout.] - [Ответ починил. Про Timeout - точно, этот ответ годится только для N = 16 ;)]

Тема 5

(См. Лекция 5: "Распределенные файловые системы")

1. Какие принципиальные решения приходится принимать при обеспечении файлового сервиса?

Решение.

1. Как определить файл - произвольная последовательность байтов + множество атрибутов, или последовательность записей, или еще как-то.
2. Можно ли модифицировать файлы после создания
3. Как защищать файлы (разграничивать доступ)
4. Использовать модель загрузки/разгрузки (полной пересылки файла) или удаленного доступа (частичной пересылки)

2. Интерфейс сервера директорий.

Решение. Обеспечивает операции создания и удаления директорий, именования и переименования файлов, перемещение файлов из одной директории в другую. Прозрачность именования. Две формы прозрачности именования различают - прозрачность расположения (/server/d1/f1) и прозрачность миграции (когда изменение расположения файла не требует изменения имени). Имеются три подхода к именованию:

1. машина + путь
2. монтирование удаленных файловых систем в локальную иерархию файлов
3. единственное пространство имен, которое выглядит одинаково на всех машинах

Двухуровневое именование. Используют большинство систем. Файлы имеют символические имена для пользователей, но могут также иметь внутренние двоичные имена для использования самой системой. Например, в операции открыть файл - символическое имя, а в ответ получает двоичное имя, которое и использует во всех других операциях с данным файлом. Способы формирования двоичных имен различаются в разных системах:

- если имеется несколько не ссылающихся друг на друга серверов (директории не содержат ссылок на объекты других серверов), то двоичное имя может быть то же самое, что и в ОС UNIX
- имя может указывать на сервер и файл
- в качестве двоичных имен при просмотре символьных имен возвращаются мандаты, содержащие помимо прав доступа либо физический номер машины с сервером, либо сетевой адрес сервера, а также номер файла.

В ответ на символьное имя некоторые системы могут возвращать несколько двоичных имен (для файла и его дублей), что позволяет повысить надежность работы с файлом.

3. Семантика разделения файлов.

Решение.

UNIX-семантика - естественная семантика однопроцессорной ЭВМ - если за операцией записи следует чтение, то результат определяется последней из предшествующих операций записи. В распределенной системе такой семантики достичь легко только в том случае, когда имеется один файл-сервер, а клиенты не имеют кэшей. При наличии кэшей семантика нарушается. Надо либо сразу все изменения в кэшах отражать в файлах, либо менять семантику разделения файлов.

Еще одна проблема - трудно сохранить семантику общего указателя файла (в UNIX он общий для открывшего файл процесса и его дочерних процессов) - для процессов на разных ЭВМ трудно иметь общий указатель.

Неизменяемые файлы - очень радикальный подход к изменению семантики разделения файлов. Только две операции - создать и читать. Можно заменить новым файлом старый - т.е. можно менять директории. Если один процесс читает файл, а другой его подменяет, то можно позволить первому процессу доработать со старым файлом в то время, как другие процессы могут уже работать с новым.

Семантика сессий - изменения открытого файла видны только тому процессу (или машине), который производит эти изменения, а лишь после закрытия файла становятся видны другим процессам (или машинам). Что происходит, если два процесса одновременно работали с одним файлом - либо результат будет определяться процессом, последним закрывшим файл, либо можно только утверждать, что один из двух вариантов файла станет текущим.

Транзакции - процесс выдает операцию НАЧАЛО ТРАНЗАКЦИИ, сообщая тем самым, что последующие операции должны выполняться без вмешательства других процессов. Затем выдает последовательность чтений и записей, заканчивающуюся операцией КОНЕЦ ТРАНЗАКЦИИ. Если несколько транзакций стартуют в одно и то же время, то система гарантирует, что результат будет таким, каким бы он был в случае последовательного выполнения транзакций (в неопределенном порядке). Пример - банковские операции.

4. Серверы с состоянием и без состояния. Достоинства и недостатки.

Решение.

Серверы с состоянием. Достоинства.

- Короче сообщения (двоичные имена используют таблицу открытых файлов).
- выше эффективность (информация об открытых файлах может храниться в оперативной памяти).
- блоки информации могут читаться с упреждением.
- убедиться в достоверности запроса легче, если есть состояние (например, хранить номер последнего запроса).
- возможна операция захвата файла.

Серверы без состояния. Достоинства.

- устойчивость к ошибкам.
- не требуется операций ОТКРЫТЬ/ЗАКРЫТЬ.
- не требуется память для таблиц.
- нет ограничений на число открытых файлов.
- нет проблем при крахе клиента.

5. Алгоритмы обеспечения консистентности кэшей в распределенных файловых системах.

Решение.

Алгоритм со сквозной записью. Необходимость проверки, не устарела ли информация в кэше. Запись вызывает коммуникационные расходы (MS-DOS).

Алгоритм с отложенной записью. Через регулярные промежутки времени все модифицированные блоки пишутся в файл. Эффективность выше, но семантика непонятная пользователю (UNIX).

Алгоритм записи в файл при закрытии файла. Реализует семантику сессий. Не намного хуже случая, когда два процесса на одной ЭВМ открывают файл, читают его, модифицируют в своей памяти и пишут назад в файл.

Алгоритм централизованного управления. Можно выдержать семантику UNIX, но не эффективно, ненадежно, и плохо масштабируется.

6. Способы организации размножения файлов и коррекции копий.

Решение. Система может предоставлять такой сервис, как поддержание для указанных файлов нескольких копий на различных серверах. Главные цели:

1. Повысить надежность.
2. Повысить доступность (крах одного сервера не вызывает недоступность размноженных файлов.
3. Распределить нагрузку на несколько серверов.

Схема реализации:

1. Явное размножение (непрозрачно). В ответ на открытие файла пользователю выдаются несколько двоичных имен, которые он должен использовать для явного дублирования операций с файлами.
2. Ленивое размножение. Одна копия создается на одном сервере, а затем он сам автоматически создает (в свободное время) дополнительные копии и обеспечивает их поддержание.
3. Симметричное размножение. Все операции одновременно вызываются в нескольких серверах и одновременно выполняются.

Протоколы коррекции.
Просто посылка сообщений с операцией коррекции каждой копии является не очень хорошим решением, поскольку в случае аварий некоторые копии могут остаться не скорректированными. Имеются два алгоритма, которые решают эту проблему.

1. Метод размножения главной копии. Один сервер объявляется главным, а остальные - подчиненными. Все изменения файла посылаются главному серверу. Он сначала корректирует свою локальную копию, а затем рассылает подчиненным серверам указания о коррекции. Чтение файла может выполнять любой сервер. Для защиты от краха главного сервера до завершения всех коррекций, до выполнения коррекции главной копии главный сервер запоминает в стабильной памяти задание на коррекцию. Слабость - выход из строя главного сервера не позволяет выполнять коррекции.
2. Метод одновременной коррекции всех копий. Все изменения файла посылаются (используя надежные и неделимые широковещательные рассылки) всем серверам. Чтение файла может выполнять любой сервер
3. Метод голосования. Идея - запрашивать чтение и запись файла у многих серверов (запись - у всех!). Запрос может получить одобрение у половины серверов плюс один. При этом должно быть согласие относительно номера текущей версии файла. Этот номер увеличивается на единицу с каждой коррекцией файла. Можно использовать различные значения для кворума чтения (Nr) и кворума записи (Nw). При этом должно выполняться соотношение Nr+Nw>N. Поскольку чтение является более частой операцией, то естественно взять Nr=1. Однако в этом случае для кворума записи потребуются все серверы.

Тема 6

(См. Лекция 6: "Распределенная разделяемая память")

Комментарии: насколько я понял, алгоритм DSM (например, с полным размножением) для поддержки определенной схемы консистентности, модифицируется. На это и основаны все задачки.

1. Какие модели консистентности памяти удовлетворяют алгоритму Деккера (алгоритм без каких-либо изменений будет работать правильно), а какие нет? Объясните ответ

Теория. Алгоритм Деккера (1968).

```
int turn;
boolean  flag[2 ];

proc( int i )
{
  while (TRUE)
  {
    <вычисления>;
    enter_region( i );
    <критический интервал>;
    leave_region( i );
  }
}

void enter_region( int i )
{
  try:  flag[i] = TRUE;
  while (flag [(i + 1) % 2])
  {
    if ( turn == i ) continue;
    flag[ i ] = FALSE;
    while ( turn != i );
    goto try;
  }
}

void leave_region( int i )
{
  turn = ( turn +1 )  % 2;
  flag[ i ] = FALSE;
}

turn = 0;
flag[ 0 ] = FALSE;
flag[ 1 ] = FALSE;
proc( 0 ) AND proc( 1 )  /* запустили 2 процесса */
```

Решение.

- Последовательная консистентность удовлетворяет. Если посмотреть на ее определение (см. след. задачу), то ясно что точно те же самые условия мы имеем и на локальном процессоре (операторы выполняются вперемешку из разных процессов, но в нужном порядке относительно каждого процесса).
- Причинная консистентность удовлетворяет, поскольку она аналогична последовательной, если сделать все управляющие переменные причинно-зависимыми.
- PRAM консистентность не удовлетворяет (см. пример в лекциях про kill), т.к. процессы могут одновременно войти в Критическую Секцию
- Процессорная консистентность удовлетворяет, т.к. есть когерентность в памяти (согласие среди процессов относительно порядка записи в каждую переменную)
- Слабая консистентность, к. по входу и по выходу не удовлетворяют. Нет работы с синхронизационными переменными, поэтому нет никакой согласованности вообще.

2. Какие модели консистентности памяти удовлетворяют алгоритму Петерсона (алгоритм без каких-либо изменений будет работать правильно), а какие нет? Объясните ответ.

Теория. Алгоритм Петерсона (1981).

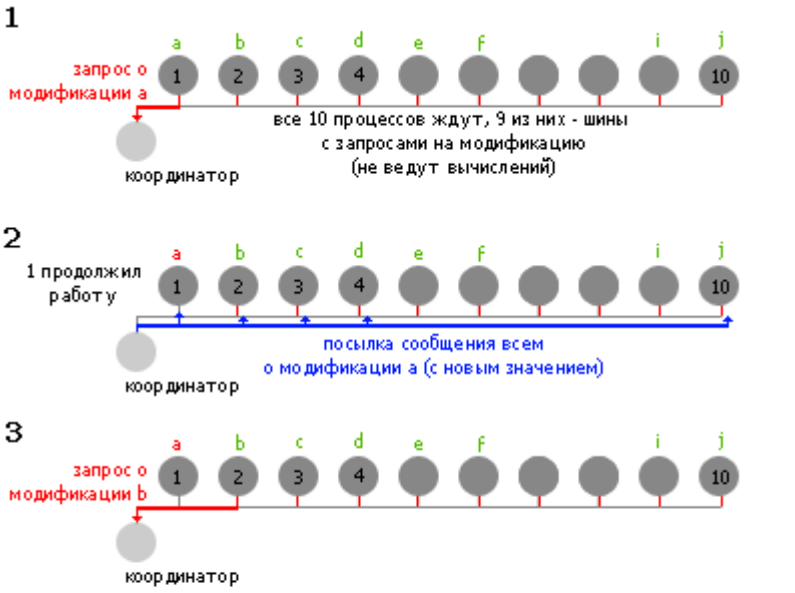
```
int turn;
int flag[ 2 ];

void enter_region( int i )
{
  int other;   /* номер другого процесса */

  other = 1 - i;
  flag[ i ] = TRUE;
  turn = i;
  while (turn == i  &&  flag[ other ] ==  TRUE) /* пустой оператор */;
}

void leave_region( int i )
{
  flag[ i ] =  FALSE;
}
```

Решение. По-моему - то же самое, что и в предыдущей задаче.

3. 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.

Теория. Последовательна консистентность. По его определению, модель последовательной консистентности памяти должна удовлетворять следующему условию: 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

Решение.



И так далее. Эта схема описавает модель консистентности, в которой каждый процесс видит глобальную последовательность операций записи.
Если бы вслед за запросом о модификации a пришел бы запрос о модификации b, то координатор присвоил бы второму запросу следующий порядковый номер и разослал бы уведомление об этой модификации вслед за уведомлением о модификации a

Допустим, что каждый процесс имеет все 10 переменных, всего процессов - N. 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 (N передач подтверждения модификации каждой размера B)

Всего у нас будет N таких запросов, а поскольку используется шина и все эти соединения должны в любом случае состоятся, то получаем общее время как сумму времен каждой передачи: N * ((N + 1) * Ts + (A + N * B) * Tb)

Ответ: N * ((N + 1) * Ts + (A + N * B) * Tb)

4. 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 запроса (при одновременных запросах - 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. "*Причинная модель консистентности памяти определяется следующим условием: Последовательность операций записи, которые потенциально причинно зависимы, должна наблюдаться всеми процессами системы одинаково, параллельные операции записи могут наблюдаться разными узлами в разном порядке."*

Из условия (см. конец задачи) ясно, что процессы не знают о причинных зависимостях между переменными. Поэтому эта задача сводится к предыдущей.

5. Процессорная 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение. *"PRAM (Pipelined RAM) консистентность определяется следующим образом: Операции записи, выполняемые одним процессором, видны всем остальным процессорам в том порядке, в каком они выполнялись, но операции записи, выполняемые разными процессорами, могут быть видны в произвольном порядке.*

Модель процессорной консистентности отличается от модели PRAM консистентности тем, что в ней дополнительно требуется когерентность памяти: Для каждой переменной x есть общее согласие относительно порядка, в котором процессоры модифицируют эту переменную, операции записи в разные переменные - параллельны. Таким образом, к упорядочиванию записей каждого процессора добавляется упорядочивание записей в переменные или группы "

Либо я чего-то не понимаю, либо ответ тот же самый, что и в предыдущих задачах. Просто возможны немного разные последовательности посылки сигналов и содержимое сообщений различается - т.е. варьируются A и B.

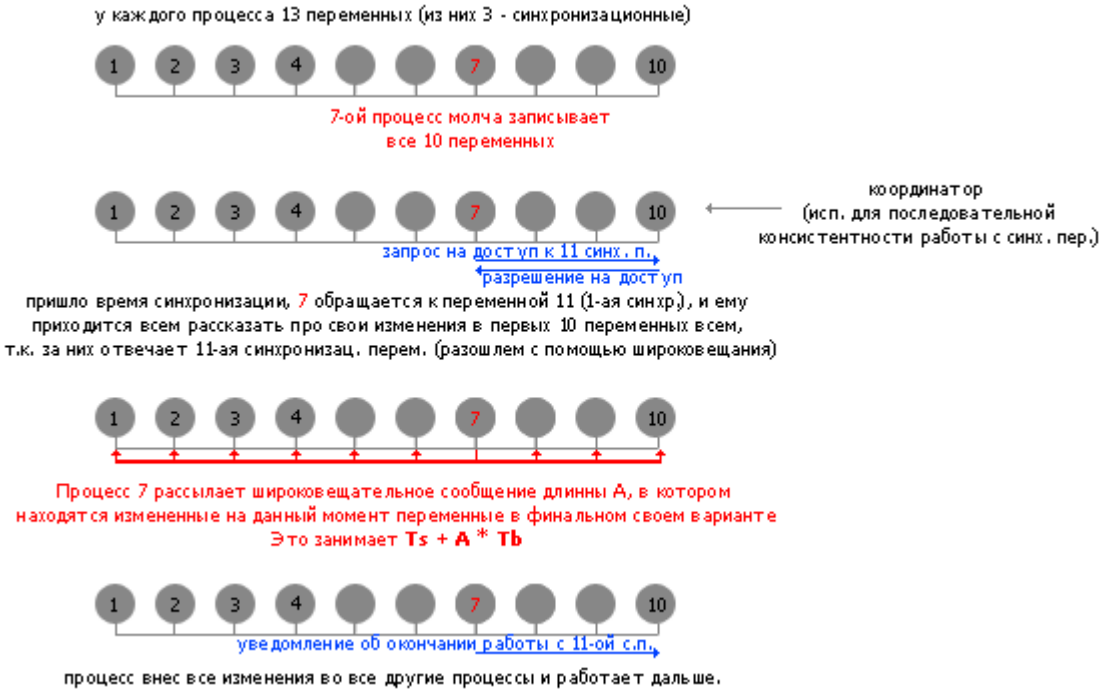
6. PRAM консистентность памяти и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует 3-кратная модификация 10 различных переменных, если все 10 процессов (каждый процесс 3 раза модифицирует одну переменную), находящихся на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания), одновременно выдали 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. *"PRAM (Pipelined RAM) консистентность определяется следующим образом: Операции записи, выполняемые одним процессором, видны всем остальным процессорам в том порядке, в каком они выполнялись, но операции записи, выполняемые разными процессорами, могут быть видны в произвольном порядке."*

То же самое. Разница может быть только в том, когда было послано какое-то подтверждение и сколько ждал какой-то процесс. 3-х кратная модификация - просто умножаем ответ на 3, все равно нужно посылать все сообщения.

7. Слабая консистентность памяти и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует модификация одним процессом 10 обычных переменных, а затем 3-х различных синхронизационных переменных, 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение. Это может быть неверно.



Пусть длинна сообщения при работе с синхр. перем. равна B. Мы получили время выполнения 3 * (Ts + B * Tb) + (Ts + A * Tb) .Если принять, что синхронизационные переменные отвечают каждая за несколько обычных переменных, то таких цикла потребовалось бы 3 (размер сообщения при широковещани A стал бы другим), то получим 3 * [3 * (Ts + B * Tb) + (Ts + A * Tb)].

Ответ: 3 * [3 * (Ts + B * Tb) + (Ts + A * Tb)]

Из другого места, описание алгоритма:

а) при модификации обычных данных записываются в локальную память.
при модификации синхронизационных переменных все модификации данных посылаются координатору, который номерует модификации, рассылает модификации широковещательно и возвращает посылающему номер последней принятой модификации. Если отправитель не имеет какой-либо записи, он должен её потребовать.

б) при чтении обычных данных они берутся из локальной памяти
при чтении синхр. переменных обращение к координатору происходит так же, как при записи.

в) значения модифицируемых переменных рассылаются координатору после обращения к синхр. перем. и далее координатором либо при обработке такого обращения, либо когда требуют пропущенные фрагменты модификаций.

г) процесс блокируется при обращении к синхр. переменным;

8. Консистентность памяти по выходу и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует трехкратное выполнение критической секции и модификация в ней 10 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.

9. Консистентность памяти по входу и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует трехкратное выполнение критической секции и модификация в ней 10 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

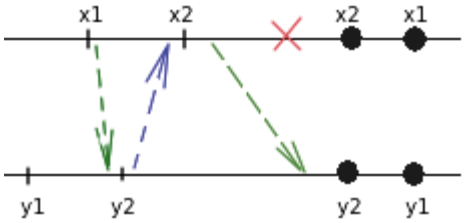
Решение.

Тема 7

(См. Лекция 7: "Отказоустойчивость")

1. Проблемы бесконечного восстановления и потери сообщений. Какие методы их решения существуют? Дайте оценку накладных расходов для сети из 10 ЭВМ с шинной организацией (без аппаратных возможностей 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.

Решение.



Сломался x и откатился на x2. Y получил сообщение-призрак, потому что в состоянии x2 X его еще не посылал, и откатился на y2. Теперь B x содержится информация о сообщении-призраке и он откатывается на x1. (Если я все правильно понимаю, нам все равно, синее сообщение принято до или после x2).
Методы решения: 1. простой консистентный метод (по локальным точкам после отправки, отправка и фиксация неделимы. Получается просто консистентное множество контрольных точек.) 2. Синхронная фиксация контрольных точек (получается строго консистентное множество).
Задачка: Поскольку синхронная фиксация в следующем вопросе, рассмотрим простой консистентный метод. Считается, что сохранение точки происходит мгновенно, тогда расход - это перепосылка в случае отката. Т.е. Для каждого отката накладные расходы складываются из пересылки "квитанций" (см. Лекции) + пересылки самих сообщений. Для одного сломавшегося процесса из 10-ти и каждого из n сообщений длиной Li байт и квитанции в 1 байт это
Sum(i=1 to n){Li + 1}
какой-то бред

2. Консистентное множество контрольных точек и алгоритмы их фиксации. Дайте оценку накладных расходов на синхронную фиксацию консистентного множества контрольных точек для сети из 10 ЭВМ с шинной организацией (без аппаратных возможностей 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 запроса (при одновременных запросах - в порядке номеров ЭВМ). Операции с файлами и процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

Решение. Алгоритмы - простой консистентный, синхронный и асинхронный
Задачка: Пусть первый - инициатор (алгоритм см. 7.1.5 в лекциях).
1-ый стартует ко 2-му и пересылает один байт. Структура сообщений такова, что одного байта на это сообщение достаточно. (100 + 1)
1-ый стартует к 3-му....
Итого: на начало первой фазы нужно 909=(Ts + Tb)*9 времени
Теперь ждем 9 ответов - столько же времени.
Теперь проводим третью фазу - тоже столько же
Итого (Ts + Tb)*9*3 = 2727

3. Протоколы голосования. Алгоритмы и применение. Дайте оценку времени выполнения одним процессом 2-х операций записи и 10 операций чтения одного байта информации с файлом, размноженным на остальных 10 ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания). Определите оптимальные значения кворума чтения и кворума записи. 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). Операции с файлами и процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

Про алгоритмы см. в лекциях (раздел "Алгоритмы голосования").
Рассмотрим механизм статического распределения голосов (еще можно оценивать вес голоса каждого из серверов в зависимости от его надежности, либо использовать динамическое распределение голосов/изменение состава голосующих (кстати, как это работает?))
Положим Vw = 10, Vr = 1 (Vw > N/2, Vr+Vw>N, а Vr удобно выбирать поменьше, потому что операций чтения много)
При модификации файла сервер сначала посылает 10 запросов на запись размером в 1 байт (тип сообщения), затем получает Vw ответов, содержащих номер последней версии файла (отведем под него 4 байта + 1 для типа сообщения).
Внимание: мы считаем, что локальная копия файла актуальна (номер версии равен максимальному из полученных номеров). Если это не так, придется обновить копию, передавая по сети файл целиком, а его размера в задаче нет. Кроме того, подразумевается, что в момент рассылки запросов никакой другой сервер не начнет делать то же самое - иначе придется учитывать таймауты и повторять рассылку еще раз.
После этого сервер модифицирует файл и рассылает остальным данные об изменениях (6 байт: собственно изменения + какая-нибудь информация о позиции в тексте + тип сообщения).
Продолжительность операции записи: ((100+1) + (100+5) + (100+6)) * 10 = 3120
При чтении данных сервер посылает запросы Vr другим серверам (5 байт - тип сообщения и информация о позиции в тексте) и получает Vr ответов (6 байт - тип сообщения, номер версии файла и данные), а затем выбирает наиболее актуальный.
Продолжительность операции чтения: ((100+5) + (100+6)) * 1 = 211
Ответ: 3120*2 + 211*10 = 8350
Желающие могут поэкспериментировать с Vr и Vw, но это приведет к увеличению времени
Примечание: в задаче написано про десять остальных серверов. Так что я предполагал, что всего их 11.

4. 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). 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. Первый процесс массово рассылает сообщение M остальным.
1) В сообщении хранится n=9 байт номеров + k=1 полезных байт. (k взято с потолка. Но так велели на консультации)
2) Это сообщение уходит n=9 раз, для каждого раза разгоняемся => время 1 фазы =
(Ts + (n+k)*Tb)*n
3) Приходит n=9 ответов по Lt = 1 байт с временными (приоритетными) метками => время =
(Ts + Lt*Tb)*n
4) Процесс-отправитель рассылает максимальную метку (опять длина = Lt) => время =
(Ts + Lt*Tb)*n
Суммируя, получаем **ответ:** n*(3*Ts + (n + k + 2*Lt)*Tb) = 2808