Распределённые системы

Лекторы: Крюков Виктор Алексеевич, Бахтин Владимир Александрович

Док собрали Синельников И., Клюшкина Е.

Спасибо всем, кто помог с составлением этого дока!
При большом количестве народа док может лагать, поэтому скачайте его перед экзаменом (Файл -> Скачать как).
Если при ответе возникнут замечания к решениям, пишите, пожалуйста, комментарии. И заносите дополнительные вопросы, если будут.

Ссылки

http://sp.cmc.msu.ru/courses/os/

Всё, что нужно: https://goo.gl/Cjhq9G

Лекции в одном файле: https://goo.gl/4nBWKH

Модели консистентности (кратко): https://goo.gl/9a0REh

Альтернативный док (NSFW на стр. 12): https://goo.gl/UovBS9

esyr.org: esyr.org wiki

Материалы в гугл диске: https://goo.gl/CjbVtw

Процедура экзамена

1. В билете 2 вопроса. На подготовку 1 час.
   Далее отвечаете на билет, получаете дополнительные вопросы.

2. Вопросы 2014 года актуальны.

3. Можно пользоваться любыми материалами, но только при подготовке к ответу.

Тема-1

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

Требуются: защита памяти, прерывания, привилегированный режим, таймер.

Если нет таких механизмов, то можно использовать виртуальную машину - единый интерпретатор, исполняющий программы пользователей и обеспечивающий эмуляцию всех этих механизмов.

Если отсутствует только один из перечисленных механизмов, можно обойтись без полной эмуляции. Например, защиту памяти будет осуществлять компилятор, причём он может проверять не всякое обращение, а один раз проверить корректность какого-либо индекса или указателя на область памяти, а затем использовать его без проверок.

Другой пример: если нет таймера, вместо полной эмуляции и увеличения счётчика при каждой команде можно увеличивать счётчик сразу на несколько единиц (например, на длину цикла).

Тема-2

При решении задач по теме 2 обратить внимание на следующее.

1. Нельзя модифицировать общие переменные вне КС.
2. При наличии вложенных КС или запросов нескольких семафоров необходимо убедиться, что не могут возникнуть тупики.
3. Нельзя обращаться к семафорам и событиям обычными операторами – только посредством операций, которые определены над ними (P, V, POST, WAIT, CLEAR).
4. Нельзя освобождать свободный семафор и объявлять уже объявленное событие.
5. Определять начальные значения семафоров и событий, если они должны быть отличны от нуля (семафор занят, событие не объявлено).

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

Вариант 1

```
struct DSemaphore {
private:
   int s;
public:
   BSemaphore access;
   BSemaphore wait;
   DSemaphore(semName_t name, int N) {     // создание владельцем семафора
         s = N;
         access = 1;
         wait = 1;
   }
   DSemaphore(semName_t name) {}           // Присоединение к семафору
   void P();
   void V();
```

```
};

void DSemaphore::P() {
  P(wait);
  P(access);
  s--;
  if (s>0)
        V(wait);
  V(access);
}

void DSemaphore::V() {
  P(access);
  s++;
  if (s==1)
        V(wait);
  V(access);
}
```

Вариант 2

```
Int S = N;
Semaphore access = 1;      // семафор для монопольного доступа к S
Semaphore wait = 1;  // при помощи него мы будем реализовывать ожидание

P(int S) {
  wait.P();
  access.P();
  S--;
  if (S > 0)         // если мы последним вошли в критическую секцию
        wait.V();    // (S == 0) - залочили после себя всех
  access.V();
}

V(int S) {
  access.P();
  S++;
  if (S == 1)        // мы освобождаем единственное место - надо разлочить
        wait.V();    // ожидающих, если мы освобождаем второе и далее место
                     // - значит очереди нет, никого разлочивать не надо
  access.V();
}
```

2. Имеется команда TSL и команда объявления прерывания указанному процессору. Опираясь на них, реализуйте на мультипроцессоре P-операцию и V-операцию для двоичного семафора. Активное ожидание освобождения семафора не допускается.

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 значениями семафоров. Для этого в начале операционные системы договариваются (командой TSL).

При освобождении семафора разблокируется первый процесс из очереди ожидающих. Это происходит на том процессоре, на котором выполнена операция освобождения, и надо известить о разблокировании все другие процессоры (или один

конкретный, на котором должен выполняться разблокированный процесс) – для этого команда прерывания.

Привязан ли процесс к процессору? Привязка позволяет повторно использовать кэш процесса, сохранённый на “его” процессоре, однако при этом появляется опасность простоя свободного процессора, не предназначенного для разблокированного процесса.

Подводя итог: прерывание необходимо при освобождении 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>
      Ret
```

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

Основное отличие событий от семафоров в том, что при объявлении одного события разблокируются сразу несколько ожидающих его процессов, ресурс не исчерпывается.

Суть операции Post: освободить семафор (V).

Суть операции Wait: запросить семафор; нам его дадут; но нужно также освободить его для других ожидающих процессов (P + V).

Начальное состояние: семафор захвачен.

```
Semaphore event;  //Должен быть в захваченном состоянии

Post(Semaphore event) {
      V(event);
}

Wait(Semaphore event) {
      P(event);
      V(event);
}
```

3б. Оцените, во сколько раз нижеприведенный алгоритм метода последовательной верхней релаксации 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 ];
semaphore s[ L1 ][ L2 ];   // массив двоичных семафоров с нулевым начальным
                           // значением
```

```
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 ]);
      }
```

Важно объявить порядок распределения витков цикла между процессами.
Пусть 1-ый процесс обрабатывает витки 1, N+1, 2N + 1, …
2-ой процесс - витки 2, N+2, 2N+2, …
т.е. распределение круговое.
Время последовательного выполнения программы:
T_L=(L1-2)*(L2-2)
Рассмотрим время выполнения параллельной программы. Если будем рассматривать все случаи, то решение получится громоздким, поэтому ограничимся следующим случаем:
$N \le L1-2$; $N \le L2-2$; (L1-2)modN=0;

Первый шаг – 1 присваивание;
Второй шаг – 2 присваивания;
……..
N-ый шаг – N присваиваний.

Т.е. первые N-1 шага работают не все процессоры. Аналогично последние N-1 шага то же самое. Первые N-1 шага назовем временем разгона. Последние N-1 шага временем торможения. За время разгона обработаем N*(N-1)/2 ячеек. За время торможения столько же. Разгон(торможение) занимает T_A=(N-1) времени.
За разгон+торможение обработаем N*(N-1) ячеек и это займет 2*(N-1) времени.
В оставшееся время работают все процессоры. Назовем это время полного параллелизма. Осталось обработать (L1-2)*(L2-2)-N*(N-1) ячеек.
Время полного параллелизма T_{FP} = ((L1-2)*(L2-2)-N*(N-1)) / N;
Время параллельной программы:
$T_P = T_{FP}+2*T_A$.
Ответ:
Ускорение $A = T_L/T_P$ = (L1-2)*(L2-2)/(((L1-2)*(L2-2)-N*(N-1)) / N + 2*(N-1)) =
= N * (L1-2) * (L2-2) / (N*(N-1)+(L1-2)*(L2-2)).

4. **Имеется механизм двоичных семафоров. Опираясь на него, реализуйте задачу читателей и писателей (алгоритмы предоставления прав доступа процессам-читателям и процессам-писателям):**
Процесс-писатель должен получать исключительный (монопольный) доступ к базе данных (других писателей или каких-либо читателей быть не должно). Произвольное число процессов-читателей может работать одновременно, но любой читатель может получить доступ только при отсутствии работающих писателей.
Запросы на доступ должны удовлетворяться “справедливо” – в порядке их поступления (можно исходить из “справедливости“ удовлетворения запросов на двоичные семафоры).

```
semaphore reader;
semaphore writer;
semaphore access;
int rd=0;
```

```
Write_enter() {
    P(reader);
    P(writer);
}
```

```
Write_leave() {
    V(reader);
    V(writer);
}
```

```
Read_enter() {
    P(reader);
    P(access);
    rd++;
    if (rd==1)
      P(writer);
    V(access);
    V(reader);
}
```

```
Read_leave() {
    P(access);
    rd--;
    if (rd==0)
      V(writer);
    V(access);
}
```

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

Алгоритм позволяет двум потокам выполнения совместно использовать неразделяемый ресурс без возникновения конфликтов, используя только общую память для коммуникации.

```
flag[0] := false
flag[1] := false
turn := 0   // or 1
```

```
p0:
    flag[0] := true
    while flag[1]=true {
        if turn ≠ 0 {
            flag[0] := false
            while turn ≠ 0 {
            }
            flag[0] := true
        }
    }

    // critical section
    ...
    // remainder section
    turn := 1
    flag[0] := false
```

```
p1:
    flag[1] := true
    while flag[0]=true {
      if turn ≠ 1 {
          flag[1] := false
          while turn ≠ 1 {
          }
          flag[1] := true
      }
    }

    // critical section
    ...
    // remainder section
    turn := 0
    flag[1] := false
```

Модели консистентности:

Модель	
Строгая	Да, так как данные полностью согласованы.

Последовательная	Да, так как процессоры видят записи в едином порядке.
Причинная	Нет, так как процессы независимо пишут в память. Причинности между записями не имеется. Процессы могут не увидеть записи, сделанной на другом процессоре, и войти в КС.
PRAM	Нет, т.к. процессы могут одновременно войти в КС (аналогично примеру в лекции про kill, где оба процесса могут быть убиты)
Процессорная	Считая переменные flag[0] и flag[1] разными, при процессорной консистентности алгоритм работать не будет. К тому же каждый процесс пишет в свой флаг, а чужой только читает.
Слабая/ По выходу/ По входу	Нет, так как необходимо выделение синхронизационных переменных и модификация кода.

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

Алгоритм:

```
flag[0]   = 0
flag[1]   = 0
turn      = 0

P0: flag[0] = 1                          P1: flag[1] = 1
    turn = 1                                 turn = 0
    while( flag[1] && turn == 1 );           while( flag[0] && turn == 0 );
            // do nothing                            // do nothing
    // critical section                      // critical section
    ...                                      ...
    // end of critical section               // end of critical section
    flag[0] = 0                              flag[1] = 0
```

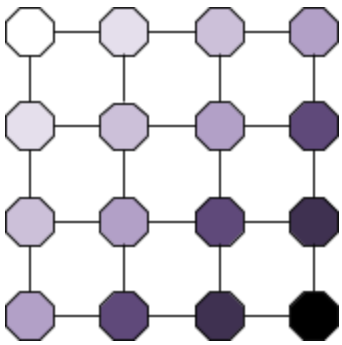
Модели консистентности:

Модель	
Строгая	Да, так как данные полностью согласованы.
Последовательная	Да. Так как все процессоры видят модификации данных в

	едином порядке (включая и автора изменений)
Причинная	Нет. Причинные связи отслеживаются по чтению, предшествующему записи. Здесь наоборот. Запись предшествует чтению, что не будет отслежено.
PRAM	Нет, так как каждый процессор сможет изменить значения разделяемых переменных в локальной памяти и войти в КС.
Процессорная	Считая переменные flag[0] и flag[1] разными, при процессорной консистентности алгоритм работать не будет, так как оба процессора могут войти в КС.
Слабая/ По выходу/ По входу	Нет, так как необходимо выделение синхронизационных переменных.

Тема-3

1. **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.**



Операция MPI_BCAST выдаётся на каждом процессе. Узнать, кто является источником сообщения, можно по аргументу root.

Без конвейера.

Операция MPI_BCAST осуществляет посылку сообщений всем соседям данного транспьютера. Следовательно, каждая посылка сообщения в транспьютерной матрице операцией (MPI_BCAST) заполняет очередную диагональ матрицы: 0 - (0,0); 1 - (1,0), (0,1); 2 - (2,0), (1,1), (0,2) и т.д. (где (i,j) - координата процесса). Следовательно, для осуществления операции MPI_BCAST в матрице 4x4 нужно 6*(Ts+N*Tb) единиц времени.

Для конвейера.

Исходим из определения времени старта:

Время старта - это время инициализации канала для передачи любого количества данных (хоть одного байта)

Разобьем сообщение на К кусков. Здесь К не может быть более 2.

Для матрицы произвольного размера n*m минимальное время передачи из (0,0) определяется числом шагов для передачи информации по самому длинному маршруту до (n, m), это (n - 1)*(m - 1) шагов.

До всех других узлов передача может быть осуществлена за меньшее число шагов.

Время передачи первого куска до последней точки (3,3) (загрузка конвейера) равно 6*(Ts+(N/K)*Tb) (см. пункт без конвейера)

Время прохода остальных кусков (конвейер загружен) (K-1)*(Ts+(N/K)*Tb)

ИТОГО: 6*(Ts+(N/K)*Tb)+(K-1)*(Ts+(N/K)*Tb)

Для особых эстетов минимум достигается при K = sqrt(5*Tb*N/Ts).

2. В транспьютерной матрице размером 4*4, в каждом узле которой находится один процесс, необходимо выполнить операцию сбора данных (длиной один байт) от всех процессов 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.

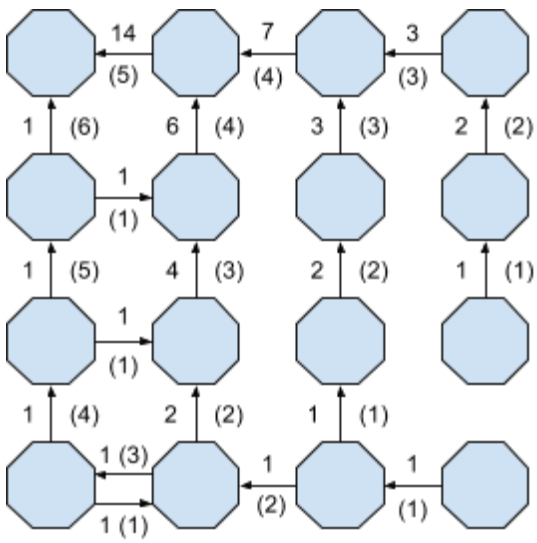
Вариант 1 (узлы не могут накапливать информацию)

2 входящих канала у собирающего узла (0,0) - можем разбить данные от всех узлов на два маршрута.

От самого удаленного узла сообщение придёт за 15 / 2 = 8 тактов.

На каждом из этих 8 тактов сообщения будут постепенно продвигаться к узлу (0,0). Таким образом за 8 тактов будут получены все сообщения. Итого: 8*(Ts+Tb).

Вариант 2 (узлы могут накапливать информацию)



Число в скобках - номер такта. Без скобок - кол-во байт.

Максимальная длина маршрута - 6 - от (3,3) к (0,0). Будем передавать по нему один

байт, не накапливая, за время 6*(Ts+Tb) = 6*(100+1) = 606.

Данные от остальных узлов будем передавать с накоплением, максимальное кол-во тактов для этого - 5. Время для трёх маршрутов длины 5:

5*Ts+(14+6+4+2+1)*Tb = 527,

5*Ts+(14+7+3+2+1)*Tb = 527,

5*Ts+(14+7+3+2+1)*Tb = 527, все меньше, чем 606 => Ответ - 606.

3. **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.**

Задача отличается от предыдущей только потоком данных. Ответ тот же.

4. **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 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Итого: 6*(Ts+Tb).

Если процессов 64, то разобьём квадрат на 4 подквадрата. Как было показано ранее, за 4 операции можно получить максимум своего квадрата в (3,3), (5,3), (3,5) и (5,5). Ещё две операции нужно на пересылку в центральные процессы. Там за 2 операции получаем максимум из всех из них (как и в первом случае), и ещё 6 на рассылку (вспомним broadcast-рассылку в квадрате 4*4). Итого: 14*(Ts+Tb).

5. **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, если передача сообщений точка-точка 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.**

Слова “очень длинное сообщение” означают, что:

1. Сообщение можно разбить на большое число частей. Тогда при использовании конвейера время его разгона и торможения будет пренебрежимо мало, то есть время не зависит от длины конвейера;
2. Каждая часть сообщения будет иметь большой размер, значит, временем Ts можно пренебречь.

Таким образом, остаётся только слагаемое L*Tb.

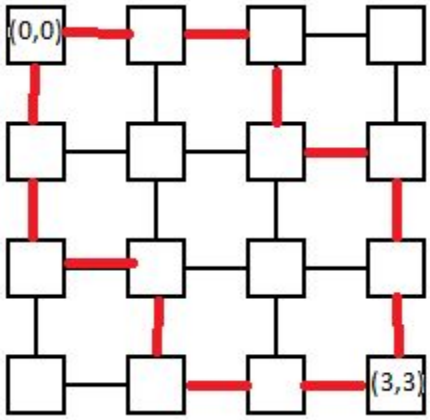
Время можно сэкономить, если задействовать два разных маршрута пересылки сообщения и конвейер.

Режимы передачи:

1. Буферизуемый. Часть сообщения помещается в буфер одного канала (по условию это копирование бесконечно быстрое), сразу после этого процесс свободен, он помещает другую часть сообщения в буфер другого канала и т.д.
2. Синхронный. Необходимо получить подтверждение готовности от принимающей стороны, общая схема: запросить готовность - получить подтверждение - начать передачу. Потеря времени.
3. Режим готовности. Считаем, что принимающая сторона заведомо готова и ждёт нашего сообщения. Самый быстрый способ, так как нет переписи в буфер и нет ожидания подтверждения от приёмника.

<u>Без конвейера.</u>

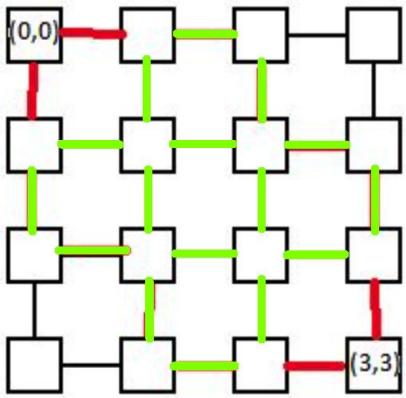
1 вариант:



6*(L/2)*Tb = 3*L*Tb.

2 вариант:

Можно также разбивать сообщения на четверти после шага 1, как на рисунке ниже (красным обозначены пересылки L/2, зелёным - L/4).



В таком случае, ответ будет (½+¼+¼+¼+¼+½)*L*Tb = 2*L*Tb.

С конвейером.

Поделим длинное сообщение на K кусков. Кроме того, его можно распилить пополам и пустить двумя путями (больше не получится – около углов узкое место), тогда время будет такое: (K+5)*((L/2)/K)*Tb.

6. **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.**

В случае с неблокирующими операциями время работы будет таким же, как с блокирующими операциями, если использовать режим передачи сообщения с буферизацией на траспьютерах.

Передавать можно с использованием конвейера или без конвейера.

Без конвейера

Потребуется 6 посылок по L/2 байт (по двум сторонам квадрата).
Время T = 6*(Ts+(L/2)*Tb).

С конвейером
Идея: разделить исходное сообщение на K частей.
Время передачи первой части - 6*(Ts+(L/K)*Tb).
Время передачи остальных частей - (K-1)(Ts+(L/K)*Tb).
Время T = (K+5)(Ts+(L/K)*Tb), K1 = sqrt(5*Tb*L/Ts), K2 = K1+1.
K = min(K1, K2).

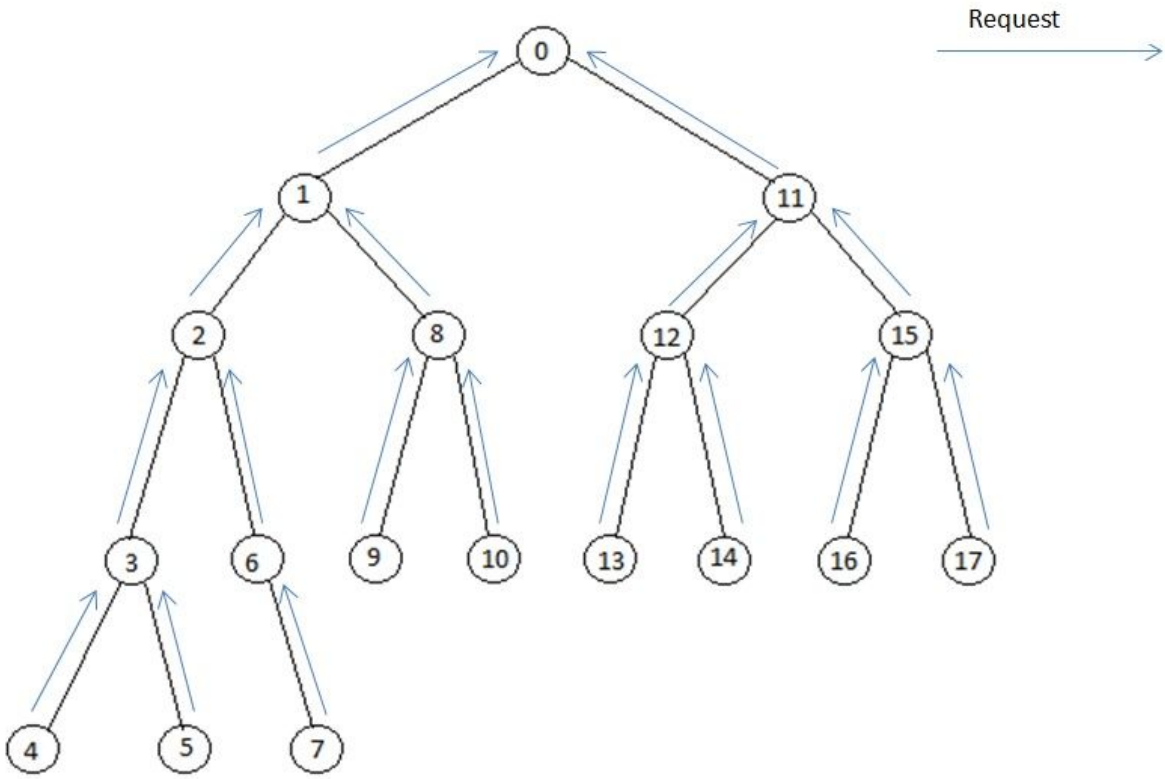
Если исходное сообщение разделить на 2, и каждое полученное сообщение, разделив на K частей, пустить по 2 параллельным маршрутам, то получим T = (K+5)(Ts+((L/2)/K)*Tb).
K1 = sqrt(5*Tb*(L/2)/Ts), K2 = K1+1.
K = min(K1, K2).

Тема-4

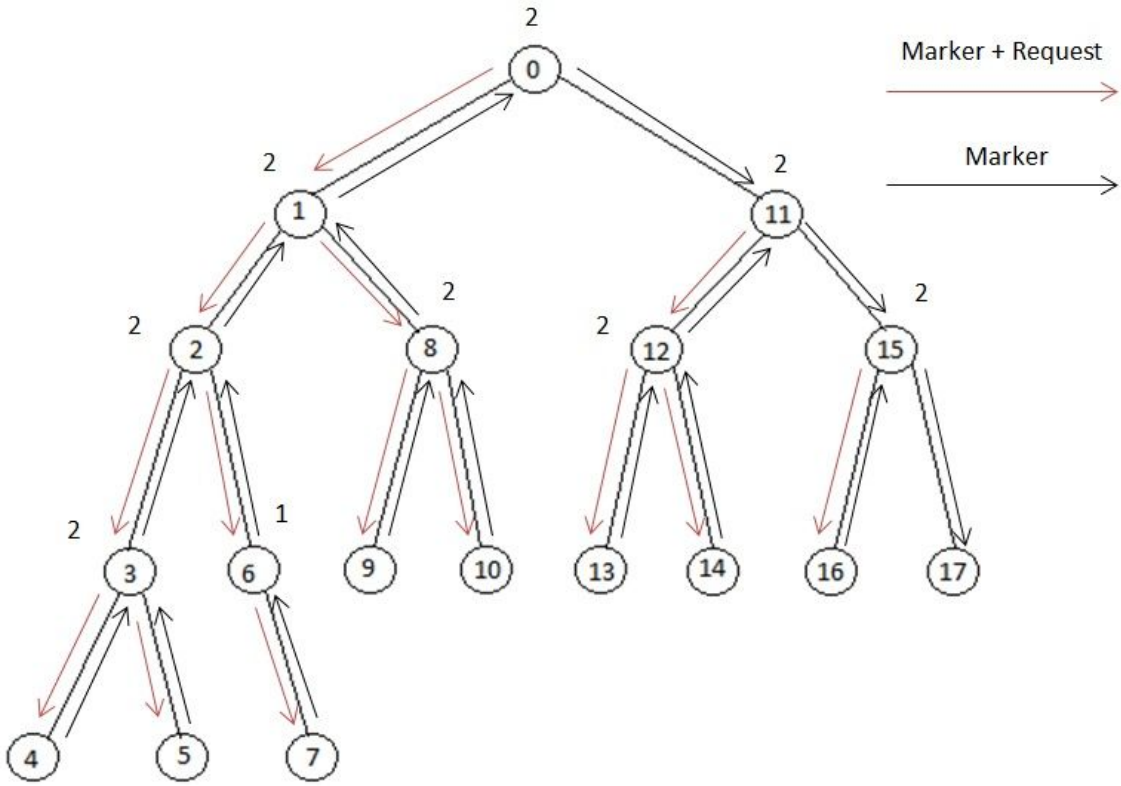
При решении задач на эту тему необходимо оговаривать нумерацию процессов в сети (связанную со структурой дерева) и начальное положение маркера.

1. **Все 16 (18) 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 (маркером владеет нулевой процесс). Время старта (время «разгона» после получения доступа к шине для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Решение для 18 процессов (число не имеет значения). Сначала будет 18 инициализирующих посылок Request.



Далее с 0-го процесса идет рассылка маркера. Главное правило: если в очереди процесса остаётся один последний запрос, то соответствующему процессу посылается только Marker. Иначе по первому в очереди запросу отправляется Marker+Request.



MR – Marker + Request.
M – Marker.
R – Request.

0—MR—1—MR—2—MR—3—MR—4—M—3—MR—5—M—3—M—2—MR—6—MR—7—M—6—M—2—M—1—MR—8—MR—9—M—8—MR—10—M—8—M—1—M—0—M—11—MR—12—MR—13—M—12—MR—14—M—12—M—11—M—15—MR—16—M—15—M—17.

Получили 17R + 14MR + 17M сообщений. Пусть Маркер и Запрос занимают по 1 байту. Маркер+Запрос занимает 2 байта.

Ответ: $T = 17*(Ts+Tb*SIZE_R) + 14*(Ts+Tb*SIZE_{MR}) + 17*(Ts+Tb*SIZE_M) = 48*Ts + 34*1 + 14*2 = 4800 + 62 = 4862$.

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 децентрализованный алгоритм с временными метками. Время старта (время «разгона» после получения доступа к шине для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

Рассмотрим каждый процесс отдельно. Он должен послать 15 запросов и получить 15 подтверждений.

В сообщении-запросе 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 T1 = 15*(Ts+3*Tb) + 15* (Ts+1*Tb).

Количество КС = 16.

Общее время T = 16*T1.

3. **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 широковещательный маркерный алгоритм (маркером владеет нулевой процесс). 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.**

Предположим, что используется режим с блокировкой процессов. То есть сообщение не может прийти, если не доставлены ранее отправленные.

Маркер у процесса 0 – он входит в критическую секцию.

Все, кроме 0-го пошлют широковещательные запросы. Это займет время 15*15*(Ts+Lзапроса*Tb).

После этого будет передаваться только маркер, содержащий очередь запросов. Всего будет 15 передач. Каждая за время Ts+Lмаркера*Tb.

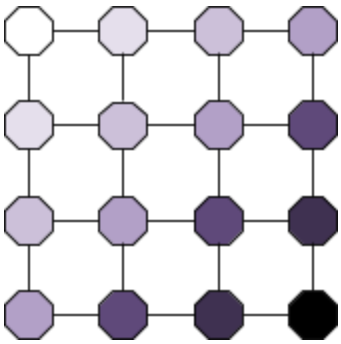
Итого 225*(Ts+Lзапроса*Tb)+15*(Ts+Lмаркера*Tb).

Замечание. Нельзя отбросить Lмаркера, так как он содержит очередь запросов и может быть достаточно большим (как минимум сравним с Ts).

4. **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.**

Как известно, для обработки КС централизованным алгоритмом нужно 3 сообщения (запрос, разрешение, подтверждение окончания).

Если считать, что **очередь запросов уже сформирована**, то с каждым процессом у координатора происходит обмен двумя сообщениями.



Мысленно пронумеруем диагонали (белый – координатор на первой диагонали).

Для каждого из 2 процессов на 2 диагонали требуется 1 передача для связи с координатором, запоминаем 2*1 = 2;

для каждого из 3 процессов на 3 диагонали – 2 передачи: 3*2 = 6;

4 процесса на 4 диагонали – 3 передачи: 4*3 = 12;

3 процесса на 5 диагонали – 4 передачи: 3*4 = 12;

2 процесса на 6 диагонали – 5 передач: 2*5 = 10;

1 процесс на 7 диагонали – 6 передач: 1*6 = 6.

2 + 6 + 12 + 12 + 10 + 6 = 48.

Итого потребуется 2*48 = 96 сообщений

Пусть размер одного сообщения 1 байт (т.к. содержится только разрешение).

Тогда ответ T = 96*(Ts+1*Tb).

Если **очередь запросов не сформирована**, то можно считать, что запросы приходят последовательно, тогда 48 умножится не на 2, а на 3 сообщения (каждый процесс отправит запрос, получит разрешение, отправит подтверждение окончания).

В этом случае ответ 3*48*(Ts+1*Tb).

Можно считать, что запросы координатор собирает параллельно. Тогда время сбора запросов можно считать равным времени работы команды MPI_GATHER. Далее по первому пункту.

5. **Сколько времени потребует выбор координатора среди 16 процессов, находящихся на разных ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания), если используется алгоритм «задиры»? «Задира» расположен в узле с координатами (0,0) и имеет уникальный номер 0. Время старта (время «разгона» после получения доступа к шине для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

В алгоритме «задиры» не определяется, в каком порядке процесс, инициирующий выборы, рассылает сообщение «ВЫБОРЫ». Говорится, что он посылает сообщение всем процессам с большими чем у него номерами.

В случае, если задира (процесс с наименьшим номером) начнёт рассылку сообщения ВЫБОРЫ **в порядке возрастания номеров процессов**, то он пошлет сообщение ВЫБОРЫ всем остальным процессам и получит от всех ответ ОК. После этого все остальные процессы будут инициировать выборы (предположим, что по той же схеме по возрастанию номеров), рассылая сообщения процессам с бóльшими номерами и получая ответы. Процесс же с наибольшим номером (15) разошлет всем сообщения КООРДИНАТОР, тем самым закончив выборы. Итого: (1 + 2 + ... + 15)*(Ts + Tb * Lвыборы) + (1 + 2 + ... + 15)*(Ts + Tb * Lок) + 15*(Ts + Tb * Lкоординатор).

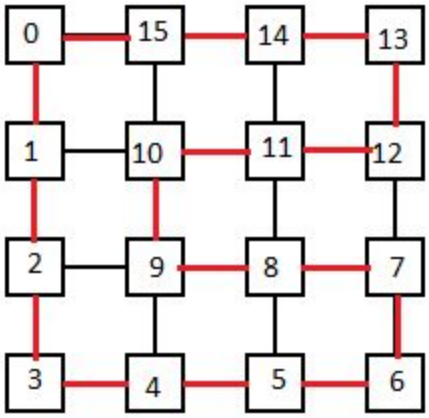
В случае, если задира **сразу выберет процесс с наибольшим номером**, он отправит ему сообщение ВЫБОРЫ, получит ОК, на этом выборы завершаются, процесс с номером 15 рассылает 15 сообщений КООРДИНАТОР.

Итого: (Ts + Tb * Lвыборы) + (Ts + Tb * Lок) + 15*(Ts + Tb * Lкоординатор).

6. **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.**

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 перенаправляет 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 1 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» + список 16 работающих процессов, и отправит процессу 1. Номер координатора отдельно не передается, им будет процесс с наибольшим номером в списке. Процесс 1 в свою очередь отправляет процессу 2, и т.д. Под «КООРДИНАТОР» отведем 1 байт для простоты. Тогда размер сообщения будет 17 байт. Для полного прохождения круга потребуется 16 сообщений. Это займет T2=16*(Ts+17*Tb).

Ответ: Общее время T=T1+T2=32*Ts+424*Tb=3624.

Тема-5

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

Файловый сервис – это то, что файловая система предоставляет своим клиентам, т.е. интерфейс с файловой системой.

- Первый фундаментальный вопрос – **что такое файл?** Большинство распределенных систем базируются на использовании среды UNIX и MS-DOS, поэтому они используют соответствующее понятие файла: это не интерпретируемая последовательность байтов. Файл может иметь *атрибуты*. Обычно файлы могут *модифицироваться* после создания, но есть и системы с неизменяемыми файлами.
- **Модель *загрузки/разгрузки* или модель *удаленного доступа*?** В первом случае файл передается между клиентом (памятью или дисками) и сервером целиком, а во втором файл сервис обеспечивает множество операций (открытие, закрытие, чтение и запись части файла, сдвиг указателя, проверку и изменение атрибутов, и т.п.). Первый подход требует большого объема памяти у клиента, затрат на перемещение ненужных частей файла. При втором подходе файловая система

функционирует на сервере, клиент может не иметь дисков и большого объема памяти.

- **Есть ли разница между клиентами и серверами?** Имеются системы, где все машины имеют одно и то же ПО и любая машина может предоставлять файловый сервис. Есть системы, в которых серверы являются обычными пользовательскими процессами и могут быть сконфигурированы для работы на одной машине с клиентами или на разных. Есть системы, в которых клиенты и серверы являются фундаментально разными машинами с точки зрения аппаратуры или ПО (требуют различных ОС, например).
- **Должны ли файловый сервер и сервер директорий быть отдельными серверами или быть объединенными в один сервер?** Разделение позволяет иметь разные серверы директорий (UNIX, MS-DOS) и один файловый сервер. Объединение позволяет сократить коммуникационные издержки.
- **Должны ли серверы хранить информацию о клиентах.** См. вопрос про сервер с состоянием.

2. Интерфейс сервера директорий.

Обеспечивает операции создания и удаления директорий, именования и переименования файлов, перемещение файлов из одной директории в другую.

Определяет алфавит и синтаксис имен. Для спецификации ***типа*** информации в файле используется часть имени (расширение) либо явный атрибут.

Все распределенные системы позволяют директориям содержать поддиректории – такая файловая система называется ***иерархической***. Некоторые системы позволяют создавать указатели или ссылки на произвольные директории, которые можно помещать в директорию. При этом можно строить не только деревья, но и произвольные графы (разница между ними очень важна для распределенных систем, поскольку в случае графа удаление связи может привести к появлению недостижимых поддеревьев. Обнаруживать такие поддеревья в распределенных системах очень трудно).

Ключевое решение при конструировании распределенной файловой системы – должны или не должны машины (или процессы) одинаково видеть иерархию директорий. Тесно связано с этим решением наличие единой корневой директории (можно иметь такую директорию с поддиректориями для каждого сервера).

Различают две формы ***прозрачности именования*** – прозрачность расположения (/server/d1/f1) и прозрачность миграции (когда изменение расположения файла не требует изменения имени).

Имеются три подхода к именованию:

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

Последний подход необходим для достижения того, чтобы распределенная система выглядела как единый компьютер.

Большинство систем используют ту или иную форму **двухуровневого именования**. Файлы (и другие объекты) имеют символические имена для пользователей, но могут также иметь внутренние двоичные имена для использования самой системой. Например, в операции открыть файл пользователь задает 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:

- 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 имён (для файла и его дублей), что позволяет повысить надежность работы с файлом.

3. Семантика разделения файлов.

- **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.

Серверы с состоянием - серверы, которые хранят информацию о клиенте (например, таблицу открытых файлов, номер последнего запроса и т.д.)

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.
- вопрос об актуальности кэша решается в момент корректировки данных, нет проблемы консистентности кэша.

Недостатки:
- накладные расходы на хранение информации об открытых файлах, на поддержание актуальности кэша.

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.

Недостатки:
- проблема захвата файла (но она решаемая).
- основная проблема – трудно работать с кэшем; если сервер знает, кому послать извещение об изменении данных, это эффективнее, иначе каждый раз при чтении должна проверяться актуальность кэша, смысл кэша вообще теряется.

5. Алгоритмы обеспечения консистентности кэшей в распределенных файловых системах.

- **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 непонятна 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 это неэффективно, ненадежно и плохо масштабируется.

6. Способы организации размножения файлов и коррекции копий.

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.

Имеются три схемы реализации размножения:

a) Явное размножение (непрозрачно). В ответ на открытие файла пользователю выдаются несколько двоичных имен, которые он должен использовать для явного дублирования операций с файлами.
b) «Ленивое» размножение. Сначала копия создается на одном сервере, а затем он сам автоматически создает (в свободное время) дополнительные копии и обеспечивает их поддержание.
c) 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 три 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 рассогласования копий в случае краха главного сервера до завершения им рассылки всех указаний о коррекции, главный сервер до выполнения коррекции своей копии 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!). Для успешного выполнения записи требуется, чтобы Nw серверов ее выполнили. При этом у всех этих серверов должно быть согласие относительно номера текущей версии файла. Этот номер увеличивается на единицу с каждой коррекцией файла. Для выполнения чтения достаточно обратиться к Nr серверам и воспользоваться одним из тех, кто имеет последнюю версию файла. Значения для кворума чтения (Nr) и кворума записи (Nw) должны удовлетворять соотношению $Nr + Nw > N$. Поскольку чтение является более частой операцией, то естественно взять $Nr = 1$. Однако в этом случае для кворума записи потребуются все серверы.

Тема-6

При ответах на вопросы по теме 6 следовать следующему плану.

1) Определение модели консистентности.
2) Алгоритм реализации в DSM с полным размножением (много писателей и много читателей, каждый из которых имеет свою копию всех переменных). Алгоритм должен быть корректным для любой коммуникационной сети и обеспечивать высокую эффективность для конкретной сети, указанной в задаче. Описание алгоритма должно содержать ответы на следующие вопросы:
    a) что делается при записи;
    b) что делается при чтении;
    c) когда, кому и как рассылаются значения модифицируемых переменных;
    d) блокируется ли процесс на время выполнения записи или рассылки значений переменных;

e) если речь идет о моделях консистентности, связанных с синхронизацией, то объяснить алгоритм синхронизации (например, алгоритм входа в КС и выхода из нее).

3) Оценить время работы описанного в пункте 2 алгоритма применительно к конкретной задаче.

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 для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

1) Результат любого действия такой же, как если бы операции (чтения и записи) всех процессов в хранилище данных выполнялись бы в некотором последовательном порядке, причём операции каждого отдельного процесса выполнялись бы в порядке, определяемом его программой.
2) 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:
  - Приписывают сообщению приоритет, помечают сообщение как «недоставленное» и буферизуют его. В качестве приоритета используется временная метка (текущее логическое время).
  - Информируют отправителя о приписанном сообщению приоритете.

2-ая фаза:

- При получении ответов от всех адресатов, отправитель:
  - Выбирает из всех приписанных сообщению приоритетов максимальный и устанавливает его в качестве окончательного приоритета сообщения.
  - Рассылает всем адресатам этот приоритет.
- Получив окончательный приоритет, получатель:
  - Приписывает сообщению этот приоритет.
  - Помечает сообщение как «доставленное».
  - Упорядочивает все буферизованные сообщения по возрастанию их приписанных приоритетов.
  - 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.

- Опрашивает всех получателей о статусе этого сообщения.
- Получатель может ответить одним из трех способов:
  - Сообщение помечено как «недоставленное» и ему приписан такой-то приоритет.
  - Сообщение помечено как «доставленное» и имеет такой-то окончательный приоритет.
  - Он не получал это сообщение.
- Получив все ответы координатор выполняет следующие действия:
  - Если сообщение у какого-то получателя помечено как «доставленное», то его окончательный приоритет рассылается всем. (Получив это сообщение каждый процесс выполняет шаги фазы 2).
  - Иначе координатор заново начинает весь протокол с фазы 1. (Повторная посылка сообщения с одинаковым приоритетом не должна вызывать коллизий).

a) При записи делается неделимая широковещательная рассылка.
b) При чтении происходит чтение из локальной памяти.
c) Значение модифицируемой переменной рассылается всем неделимо широковещательно при записи.
d) На время выполнения записи или рассылки значений переменных процесс блокируется, т.к. надо ждать завершения неделимой широковещательной рассылки.

3) Оценка времени.

Оценим время одной широковещательной рассылки.

1 фаза. Процессов у нас 10. Т.е. текущему процессу надо отправить девяти остальным процессам адрес и значение модифицируемой переменной, список идентификаторов сообщений, кому ещё 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 $T1 = Ts+11*Tb = 111$ ($T1 = Ts + 2*Tb = 102$). Отправка девяти

сообщений T2 = 9*T1 = 999 (T2 = 9*T1 = 918). Каждый из девяти процессов должен ответить отправителю приписанным приоритетом. Пусть под приоритет отведем 1 байт. Тогда все ответы займут T3 = 9*(Ts+1*Tb) = 9*101 = 909.
Первая фаза займет T4 = T2 + T3 = 999 + 909 = 1908 (T4 = T2 + T3 = 918 + 909 = 1827).

2 фаза. Отправитель рассылает максимальный приоритет девяти процессам. Это займет T5 = 9*(Ts+1*Tb) = 9*101 = 909.

Общее время обеих фаз займет T6 = T4 + T5 = 1908 + 909 = 2817 (T6 = T4 + T5 = 2736).

Это было время одной широковещательной упорядоченной рассылки. Нам нужно десять таких. Итоговое время T = 10*T6 = 28170 (T = 10*T6 = 27360).

2. Причинная консистентность памяти и алгоритм ее реализации в DSM с полным размножением при условии, что никаких сведений от компилятора о причинной зависимости операций записи не имеется. 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 для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

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) Реализация причинной консистентности может осуществляться следующим образом:
   i) все модификации переменных на каждом процессоре нумеруются;
   ii) 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;
   iii) выполнение любой модификации на каждом процессоре задерживается до тех пор, пока он не получит и не выполнит все те модификации других процессоров, о которых ему было известно.

   a) При записи модификация рассылается всем процессам;
   b) При чтении происходит чтение из локальной памяти;
   c) Модифицируемая переменная рассылается всем процессам при записи;
   d) На время выполнения записи или рассылки значений переменных процесс не блокируется;

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 – по числу процессов. 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-го процесса – 2 байта, второго – 4, третьего – 6 и т.д.

Размер десятого – 20 байт. Соответственно времена:
T1 = 9*(Ts+2*Tb)
T2 = 9*(Ts+4*Tb)
……..
T10 = 9*(Ts+20*Tb)
Общее время T = T1+….+T10 = 9*(10*Ts+110*Tb) = 9*(1000+110) = 9*1110 = 9990.

3. **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 для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

1) Процессорная консистентность – каждый процессор видит модификации, производимые одним процессором, в том же порядке, как они были произведены. Кроме того для каждой переменной x есть общее согласие относительно порядка, в котором процессоры модифицируют эту переменную, операции записи в разные переменные – параллельны.

2) Сам алгоритм. За каждую группу переменных отвечает свой координатор, который получает от процессов запросы на модификацию и рассылает всем указания о проведении модификации. Чтобы не нарушить порядок получения процессами указаний о модификациях различных переменных, запрошенных одним процессом у разных координаторов, надо каждому процессу нумеровать свои модификации, и эти номера должны рассылаться всем вместе с указаниями о проведении модификаций. Тогда любой процесс, получающий указание о проведении модификации, может задержать его выполнение до получения недостающих указаний о предшествующих модификациях соответствующего процесса.

    a) при записи координатору, который отвечает за данную переменную, отправляется запрос на модификацию;
    b) при чтении происходит чтение из локальной памяти;
    c) координатор принимает запросы на модификацию и рассылает всем указания о проведении модификации;
    d) на время выполнения записи или рассылки значений переменных процесс блокируется и ждет получения указания от координатора о модификации.

3) Оценка времени.

Пусть каждый процессор отвечает за одну переменную.

1 вариант. Пусть координатором переменной, которую изменяет данный процессор не является он сам, т.е. для каждого процессора координатором изменяемой им переменной является другой процессор. Так как в случае, если координатор изменяет свою переменную, задача сводится к простой рассылке модификаций. У нас все переменные разные, поэтому можно в них записывать параллельно. Рассмотрим время выполнения одной модификации. Сначала координатору отправляется запрос на модификацию. Он рассылает указания о проведения модификации (всем, кроме себя). В запросе на модификацию содержится адрес переменной, ее значение, номер модификации. Координатор может прикреплять к указаниям о модификации номера процессов, запросивших эти модификации, чтобы можно было различать модификации с одинаковыми номерами различных процессов. Время модификации = время запроса + время рассылки:

$T_M = T_R + 9*T_{MD}$,
M – Modification;
R – Request;
MD – ModificationDirective.

$T_R = Ts + SIZE_R*Tb$;
$T_{MD} = Ts + SIZE_{MD}*Tb$;

Для простоты, пусть под адрес переменной, значение переменной, номер модификации отведем по 1 байту. Тогда $SIZE_R$ = 3 байта. Под номер процесса отведем тоже 1 байт. $SIZE_{MD}$ = 4 байта. Так как у нас 10 модификаций, общее время:

$T = 10*T_M = 10*(T_R + 9*T_{MD}) = 10*(Ts + 3*Tb + 9*(Ts+4*Tb)) = 10*(10*Ts + 39*Tb) = 10390$.

2 вариант. Каждый из процессов модифицирует переменную, которой владеет; получается, что только отсылается сообщение о модификации остальным процессам:

$T_R = 0$;

$T = 10*T_M = 10*(T_R + 9*T_{MD}) = 90*(Ts + SIZE_{MD}*Tb) = 90*(100 + 4) = 9360$.

4. **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 для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

1) 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.

2) Алгоритм: выстраиваем операции модификации в конвейер, можем не ждать окончания операции, надо только быть уверенным, что модификацию увидят все в одном порядке.

    a) при записи – ставим сообщение о модификации в конвейер, продолжаем работу; сообщение посылается, если пришло подтверждение о предыдущей модификации (подтверждения гарантируются протоколом передачи);
    b) чтение – из локальной памяти;
    c) значения модифицируемых переменных рассылаются другим процессам при изменении (записи) переменных;
    d) блокировок нет.

3) 3 раза модифицируют переменную. 3 раза надо послать сообщение о модификации каждым процессом всем остальным. Итого 3*9*10*(Ts + Lсообщ*Tb).

5. 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 сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

1) При слабой консистентности соблюдается последовательная непротиворечивость между группами операций, а не между отдельными операциями. Для выделения этих групп используются переменные синхронизации (Таненбаум – «Распределенные системы», слабая непротиворечивость).
   Модель слабой консистентности имеет три свойства:
    1. Доступ к синхронизационным переменным определяется моделью последовательной консистентности;
    2. Доступ к синхронизационным переменным запрещен (задерживается), пока не выполнены все предыдущие операции записи;
    3. Доступ к данным (запись, чтение) запрещен, пока не выполнены все предыдущие обращения к синхронизационным переменным.

2) См. пп. e)

    a) запись происходит в локальную память;
    b) чтение происходит из локальной памяти;
    c) при вызове операции синхронизации значения модифицируемых переменных рассылаются всем;
    d) процесс, вызвавший операцию синхронизации, блокируется пока не завершится операция синхронизации;

e) Алгоритм. Процесс, вызвавший операцию синхронизации, посылает всем процессам указание о модификации переменных посредством механизма упорядоченного широковещания (аналог децентрализованного алгоритма для последовательной непротиворечивости, который приведен в лекциях, таким образом достигается последовательная консистентность для групп операций).

3) Оценка времени. При модификации переменные пишутся в локальную память. Когда вызывается первая операция синхронизации, модификация рассылается широковещательно. В сообщении содержатся адреса переменных и их значения. Не нарушая общность рассуждений, для простоты отведем под адрес и значение по 1 байту. Тогда первая синхронизация займет $T_{S1} = Ts + 10*(SIZE_{ADDRESS} + SIZE_{VALUE})*Tb = 120$. Т.к. после первой синхронизации ничего не модифицировали, можем не делать широковещательную рассылку, т.к. в силу данного алгоритма при чтении всегда имеем актуальные копии (то, которое записал какой-то процесс, успевший сообщить об этом всем посредством синхронизации памяти, а если не имели бы актуальные копии, то пришлось бы их доставать). Поэтому при вызове операции синхронизации имеет смысл делать широковещательную рассылку только при наличии модификации после предыдущего вызова операции синхронизации. Т.е. при вызове остальных синхронизации не будет никаких обменов сообщениями.

   Ответ: Общее время $T = T_{S1} = 120$.

<u>Дополнительные вопросы.</u>

1. *Как определяются переменные, которые модифицировались?*
   Модифицированные переменные можно определить двумя способами:
   a. С помощью компилятора, т.е. компилятор вместо операций присваивания может вставлять вызовы определенных функций;
   b. Запоминать старые значения переменных и потом сравнивать. В данном случае можно каждый раз сразу же после операции синхронизации обновлять значения запомненных переменных и при вызове сравнивать текущие значения переменных с запомненными. Если есть расхождение, то рассылать текущие значения переменных, если нет расхождений, то ничего делать. Т.о. перечень действий при вызове операции синхронизации:
      i. сравнить текущие значения с сохраненными;
      ii. если есть расхождения, то разослать текущие значения переменных, которые расходятся, а также обновить значения сохраненных переменных.

2. *Если при синхронизации в процессе А все измененные им переменные рассылались всем, то зачем процессу Б требуется перед чтением переменной давать операцию синхронизации?*
   Компилятор может хранить значения переменных на регистрах, в кэше и т.д., обновить их другие процессы не могут, поэтому операция синхронизации перед чтением нужна, чтобы компилятор обновил такие значения (а в его памяти они будут обновлены при выполнении операциях синхронизации, выданных другими процессами).

6. **Консистентность памяти по выходу и алгоритм ее реализации в DSM с полным размножением. Сколько времени потребует трехкратное выполнение каждым процессом критической секции, в которой модифицируются 10 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 сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

1) Консистентность по выходу - совместно используемые данные становятся непротиворечивыми после выхода из критической секции. В модели консистентности по выходу введены специальные функции обращения к синхронизационным переменным: ACQUIRE - захват синхронизационной переменной, информирует систему о входе в критическую секцию; RELEASE - освобождение синхронизационной переменной, определяет завершение критической секции. Захват и освобождение используются для организации доступа не ко всем общим переменным, а только к тем, которые защищаются данной синхронизационной переменной. Такие общие переменные называют защищенными переменными.

2) Алгоритм. Фактически, используется маркер, в котором хранится информация о модификациях. Маркер присутствует у процесса, который выполнил последние преобразования с критической секцией (или выполняет). Все преобразования заносятся в маркер. Ленивая реализация: после преобразований процесс не отсылает результаты, а передает их по требованию. Процесс, которому нужен доступ в секцию, отсылает запрос о маркере.

    a) при записи обычной (не синхронизационный) переменной – запись в локальную память;
    b) чтение – из локальной памяти;
    c) выход из критической секции – запись всех модификаций в маркер, вход в критическую секцию – запрос маркера у всех процессов, блокировка до появления маркера; если пришел запрос о маркере – запоминается;
    d) блокировка – только при входе в критическую секцию;

3) Процесс посылает всем запрос на маркер, T1 = Ts + Lзапроса*Tb (учтена возможность аппаратного широковещания). В ответ один из процессов передаёт ему маркер, T2 = Ts + Lмаркера*Tb. Это делает каждый процесс, 3 раза.
   Итого: T = 3*10*(T1 + T2) = 30*(2*Ts + (Lзапроса + Lмаркера)*Tb).

   Возможна реализация, когда каждый процесс 3 раза выполняет критическую секцию, после чего только выполняет передачу маркера. В таком случае время в 3 раза меньше.

7. **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 сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память считаются бесконечно быстрыми.**

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

2) Тут опять маркер. Реализация почти как в предыдущем алгоритме, только с каждой синхронизационной переменной связан свой маркер и в нем хранятся модификации определенных переменных (т.к. с каждой синхронизационной переменной связаны свои переменные). Если процесс хочет захватить переменную в немонопольном режиме, то ждет сообщения от владельца переменной (хотя бы одного). Если в монопольном, то нужен ответ от всех процессов (либо ок, либо маркер).

    a) запись – из локальной памяти;
    b) чтение – из локальной памяти;
    c)
        i) захват переменной в немонопольный режим – шлем всем сообщение «немонопольный захват», ждем маркер;
        ii) захват в монопольном режиме – шлем всем сообщение «монопольный захват», ждем ответа от всех;
        iii) если пришел «монопольный захват», маркера нет и он нам не нужен, то шлем «ок»;
        iv) если пришел «монопольный захват», маркера нет, но он нужен, то ждем маркер и потом отправляем «ок»;
        v) если пришел «монопольный захват» и есть маркер, шлем маркер;
        vi) если пришел «немонопольный захват», есть маркер, которым мы владеем немонопольно, то шлем копию маркера запросившему процессу;
        vii) при выходе из секции модифицируем маркер;
    d) при каждом запросе «монопольный захват» или «немонопольный захват» - блокировка до получения необходимых ответов;

3) Все модификации монопольные. 3 раза критическая секция, каждый шлет всем сообщение о запросе критической секции. Далее все отвечают (т.к. монопольная модификация), при этом будет 8 сообщений «ок» и одно с маркером. Это делает каждый процесс, 3 раза.
   T1 = Ts + Tb*Lзапроса (учтена возможность аппаратного широковещания);
   T2 = 8*(Ts + Tb*Lок);
   T3 = Ts + Tb*Lмаркера;
   Итого: 3*10*(T1 + T2 + T3) = 30*(10*Ts + (Lзапроса + 8*Lок + Lмаркера)*Tb).

В нашем случае нет необходимости в независимом параллельном доступе к переменным. Но если считать, что с каждой переменной связана своя синхронизационная переменная, то ответ увеличивается в 10 раз.

Тема-7

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 сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

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.

1. Процесс-отправитель посылает сообщение группе процессов (список их идентификаторов содержится в сообщении).
2. При получении этого сообщения процессы:
    a. Приписывают сообщению приоритет, помечают сообщение как «недоставленное» и буферизуют его. В качестве приоритета используется временная метка (текущее логическое время).
    b. 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.
3. Получив окончательный приоритет, получатель:
    a. Приписывает сообщению этот приоритет.
    b. Помечает сообщение как «доставленное».
    c. Упорядочивает все буферизованные сообщения по возрастанию их приписанных приоритетов.
    d. 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:
    - Сообщение помечено как «недоставленное» и ему приписан такой-то приоритет.
    - Сообщение помечено как «доставленное» и имеет такой-то окончательный приоритет.
    - Он не получал это сообщение.
2 Получив все ответы, координатор выполняет следующие действия:
    a. Если сообщение у какого-то получателя помечено как «доставленное», то его окончательный приоритет рассылается всем. (Получив это сообщение каждый процесс выполняет шаги фазы 2).
    b. Иначе координатор заново начинает весь протокол с фазы 1. (Повторная посылка сообщения с одинаковым приоритетом не должна вызывать коллизий).

Итого:

- 5 сообщений посылается, после чего падает отправитель
- 5 сообщений с временными метками возвращается
- тайм-аут
- 8 сообщений от какого-нибудь процесса о времени (т.к. адресатов изначально было 9, один из них рассылает сообщения остальным)
- 4 ответа о том, что не доставлено + приоритет
- 4 ответа о том, что не получал
- 8 сообщений от координатора
- 8 ответов с временными метками
- 8 сообщений с финальным временем

55*Ts + 5*req*Tb + 5*time*Tb + 8*ask*Tb + 4*ans*Tb + 4*none*Tb + 8*req*Tb + 8*time*Tb + 8*final*Tb

ВРЕМЯ СОСТОИТ ИЗ СЧЕТЧИКА СОБЫТИЙ И НОМЕРА ПРОЦЕССА!

Req – сообщения-запросы (например, 9 байт с идентификаторами, 2 байта на начальный приоритет - счетчик событий и номер процесса, 1 байт - содержательная информация в сообщении), 12 байт.

Time – сообщения с логическим временем, 2 байта.

Ask – сообщения-вопрос типа «есть ли у кого-нибудь доставленное сообщение?» (1 байт с номером сообщения, 1 байт с полезной инфой).

Ans – ответы на вопрос координатора типа “не доставлено, приоритет такой-то”, 2 байта.

None - ответы на вопрос координатора типа “не получал”, 1 байт.

Final – финальное сообщение с итоговым логическим временем, 2 байта.

2. **Протоколы голосования. Алгоритмы и применение. Дайте оценку времени выполнения одним процессом 2-х операций записи и 10 операций чтения N байтов информации с файлом, расположенным (размноженным) на остальных 10 ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания). Определите оптимальные значения кворума чтения и кворума записи для N=300. 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.**

Общая схема использования голосования при размножении файлов может быть представлена следующим образом.
Файл может модифицироваться разными процессами только последовательно (при открытии файла на запись процесс-писатель будет ждать закрытия файла другим писателем или всеми читателями), а читаться всеми одновременно (протокол писателей-читателей). Все модификации файла нумеруются, и каждая копия файла характеризуется номером версии – количеством ее модификаций. Каждой копии приписано некоторое количество голосов Vi Пусть общее количество приписанных всем копиям голосов равно V. Определяется кворум записи Vw и кворум чтения Vr так, что

$$Vw + Vr > V$$

Для записи информации в файл писатель рассылает ее всем владельцам копий файла и должен получить Vw голосов от тех, кто успешно выполнил запись.
Для получения права на чтение читателю достаточно получить необходимое число голосов (Vr) от любых серверов. Кворум чтения выбран так, что хотя бы один из тех серверов, от которых получено разрешение, является владельцем текущей копии файла. За чтением информации из файла читатель может обратиться к любому владельцу текущей копии файла.
Описанная схема базируется на статическом распределении голосов. Различие в голосах, приписанных разным серверам, позволяет учесть их особенности (надежность, эффективность). Еще большую гибкость предоставляет метод динамического перераспределения голосов.
Для того чтобы выход из строя некоторых серверов не привел к ситуации, когда невозможно получить кворум, применяется механизм изменения состава голосующих.

Различают протокол принятия единого решения и протокол принятия согласованных решений.

Хотя собственно алгоритм предписывает назначить каждой копии отдельное число голосов, будем считать, что у каждой копии ровно 1 голос.

Алгоритм:

<u>Запись:</u>

- Посылаем всем процессам запрос на модификацию длины Lreq

- Должны получить все ответы с текущими версиями файла.
- Предположим, что имеет место согласие всех серверов относительно версии файла, тогда достаточно произвести Vw записей.

Чтение:

- Посылаем запрос на чтение из К байт служебной информации Vr процессам.
- Получаем Vr ответов с текущими версиями файла.
- Выбираем наиболее свежую и читаем её.

Одна запись требует 10*(Ts + Lreq*Tb) + 10*(Ts + Lresp*Tb) + Vw*(Ts + N*Tb).
Одно чтение требует Vr*(Ts + Lreq*Tb) + Vr*(Ts + Lresp*Tb) + 1*(Ts + N*Tb).

Итого 2*[10*(Ts + Lreq*Tb) + 10*(Ts + Lresp*Tb) + Vw*(Ts + N*Tb)] + 10*[Vr*(Ts + Lreq*Tb) + Vr*(Ts + Lresp*Tb) + 1*(Ts + N*Tb)].

Если принять Lresp = Lreq = 1 байт, то необходимо минимизировать 2*(400*Vw + 2020) + 10*(202*Vr + 400) = 800*Vw + 8040 + 2020*Vr.

Наилучший результат при Vw = 10, Vr = 1.

3. **Консистентное и строго консистентные 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 T1. Время старта (время «разгона» после получения доступа к шине для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Доступ к шине ЭВМ получают последовательно в порядке выдачи запроса на передачу (при одновременных запросах – в порядке номеров ЭВМ). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.**

1. Множество контрольных точек называется строго консистентным, если во время его фиксации никаких обменов между процессами не было.
2. Множество контрольных точек называется консистентным, если для любой зафиксированной операции приема сообщения, соответствующая операция посылки также зафиксирована (нет сообщений-сирот).

Простой метод фиксации консистентного множества контрольных точек - фиксация локальной контрольной точки после каждой операции посылки сообщения.

Синхронная фиксация контрольных точек и восстановление. Требование: надежная передача сообщений (сообщения всегда приходят, схема FIFO).

Два вида точек – постоянная (уже точно установлена) и пробная – становится постоянной после окончания работы алгоритма.

Сохранение:

- процесс создает пробную точку, посылает всем уведомление

- все создают пробную точку, отвечают процессу-инициатору, при этом нельзя посылать неслужебные сообщения
- процесс шлет всем информацию о создании новой постоянной точки, пробная становится постоянной
- Оптимизация: если не было сообщений после последней контрольной точки, то можно не создавать новую.

Восстановление:

- инициатор шлет всем сообщение о подготовке к откату
- все отвечают, что готовы, после этого до конца алгоритма не посылается неслужебных сообщений
- инициатор шлет сообщение об откате, все откатываются
- Оптимизация: если не было сообщений с момента фиксации, то можно не откатываться.

Асинхронный алгоритм. Фиксация может производиться асинхронно. В этом случае множество контрольных точек может быть не консистентным. При откате происходит поиск подходящего консистентного множества путем поочередного отката каждого процесса в ту точку, в которой зафиксированы все посланные им и полученные другими сообщения (для ликвидации сообщений-сирот). Алгоритм опирается на наличие в стабильной памяти для каждого процесса журнала, отслеживающего номера посланных и полученных им сообщений, а также на некоторые предположения об организации взаимодействия процессов, необходимые для исключения «эффекта домино» (например, организация приложения по схеме сообщение-реакция-ответ).

Решение задачи:

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 тем, 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 байт. 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 сообщений было отправлено, тоже кроме себя. Такие обмены списками происходит дважды между каждым процессом и инициатором. Каждый обмен занимает T2 = Ts + 18*Tb. Всего обменов 9*2 = 18.

Ответ: Общее время T = T1 + 18*T2 = T1 + 2124.

Тема-8 (на экзамене не будет (совсем не будет))

Дополнительные задачи

1. Какую модель консистентности можно реализовать в мультипроцессорах (с общей памятью)?

Ответ:

Последовательную. Строгую не можем, так как у процессоров имеется кэш.

2. MPI. В синхронизационном режиме отправка сообщения не начинается, пока у процесса, который должен принять сообщение, не появится RECEIVE. А как мы это узнаем?

Ответ:

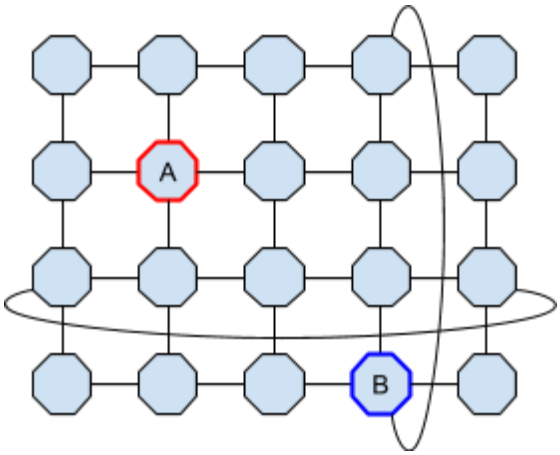
Процесс, в котором появился SEND, должен отправить запрос, есть ли на другом конце RECEIVE. Это должен сделать именно он, так как:

1. он знает получателя (второй процесс может не знать отправителя),
2. он знает, что отправка будет производиться в синхронизационном режиме (получатель не может и не должен знать режим).

3. Какую модель консистентности необходимо добавить к причинной, чтобы полученная модель оказалась не слабее процессорной консистентности?

Ответ: PRAM консистентность

4. Транспьютерная матрица 4x5:



Необходимо передать сообщение длиной N=79 байт из A в B. Сколько времени потребуется для этого, если Ts=0, Tb=1. За один такт по одному каналу можно передать 1 байт.

Решение:

Разделяем на 4 маршрута. Из левого соседа можно достичь B за 4 такта, из остальных - за 3. Влево будем передавать 19 байт сообщения, по остальным направлениям - по 20.

Время для передачи сообщения длины L на расстояние K - T=T1+T2, где T1 - время заполнения конвейера (по сути - передача первого байта), T2 - время передачи всех частей, кроме первой:

T1=K*(Ts+Tb),

T2=(L-1)*(Ts+Tb).

В нашем случае по левому направлению время T1=5, T2=18; по остальным трём - T1=4, T2=19.

Ответ: T = 23.

См. также esyr.org